M16C Family
Software driver for M32C/83's GCI and HDLC feature

1 Abstract

This application note describes the software driver, which is necessary in order to access the M32C/83 function blocks for GCI/PCM and HDLC using applications. The software driver is responsible for initialization and control of the hardware function blocks and for the setup of the data transfer between the blocks.

2 Contents

3 GCI/PCM interface

3.1 General Description of GCI/PCM

The GCI (General Circuit Interface)/PCM (Pulse Code Modulation) highway standard defines an industry-standard serial bus for interconnecting telecommunications ICs. This interface is a pure physical interface. The serial bus GCI/PCM provides a full-duplex communication link containing user control data, control/programming, and status channel. The data clock (DCL) is used to clock data and operates at twice the data rate (except for PCM mode). The frames are delimited by an 8-kHz frame sync signal (FSC). Dout/Din (data upstream/data downstream) are the up/down serial information streams. The M32C/83 supports the GCI/PCM interface in TE, NT and PCM highway mode.

3.1.1 GCI TE mode

The GCI TE mode is designed for ISDN terminal applications. The up/downstream data link consists of three channels, each containing 32 bits. This 12 bytes frame is repeated at 8kHz, headed by a rising FSC signal, giving a data rate of 768kbit/s. Note that the DCL frequency is 1.536MHz, so DCL have to be divided by the GCI interface of the M32C/83.

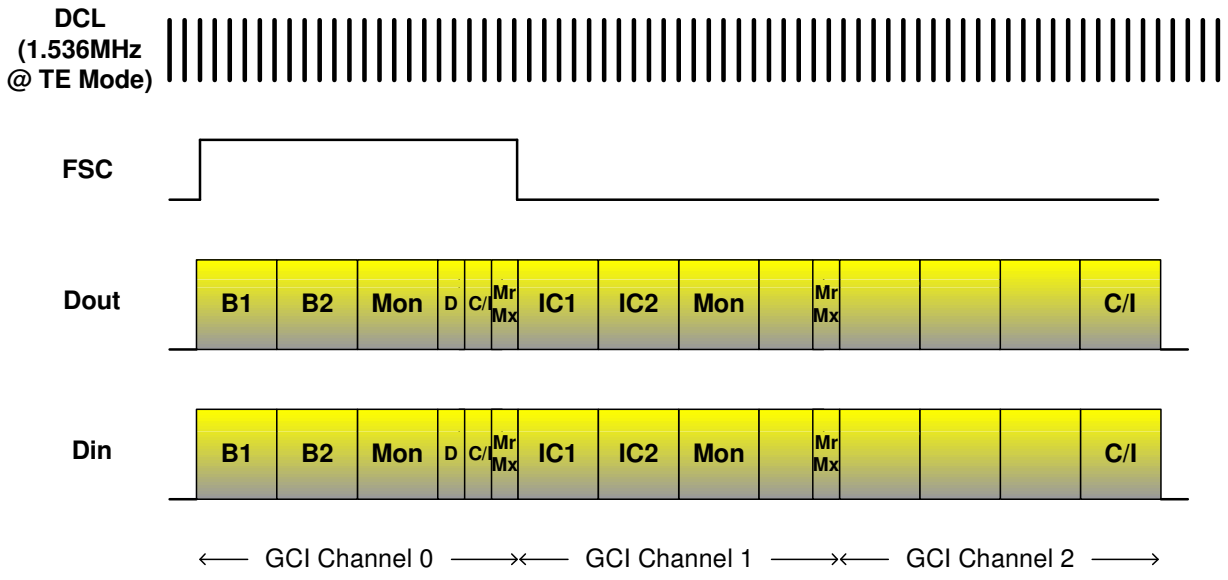


Figure 1: GCI simplified timing diagram for TE mode

TE mode (f_{DCL}= 1536kHz, f_{Data} = 768kHz)
Dout and Din are containing following data:

- B1 : Voice data or other data 8Bit
- B2 : Voice data or other data 8Bit
- Mon: Control data for layer 1 device 8Bit
- D: D channel control data 2Bit
- C/ I: IC's intercommunication (channel 0) 4Bit
- MrMx: Handshake for Mon channel 2Bit
- IC1 IC's intercommunication 8Bit
- IC2 IC's intercommunication 8Bit
- C/ I: IC's intercommunication (channel 2) 8Bit

3.1.2 GCI NT mode

The NT mode of the GCI interface provides a connection path between line transceivers (ISDN) and codecs. The M32C/83 set into NT mode can act like a switch backbone. In this mode ISDN transceiver and/or codecs/filters could be connected to the bus. Data, control and status information is multiplexed into frames, which are transmitted in an 8kHz rate. One NT frame is divided into 8 sub-frames, whereby one sub-frame is being dedicated to each transceiver or pair of codecs.

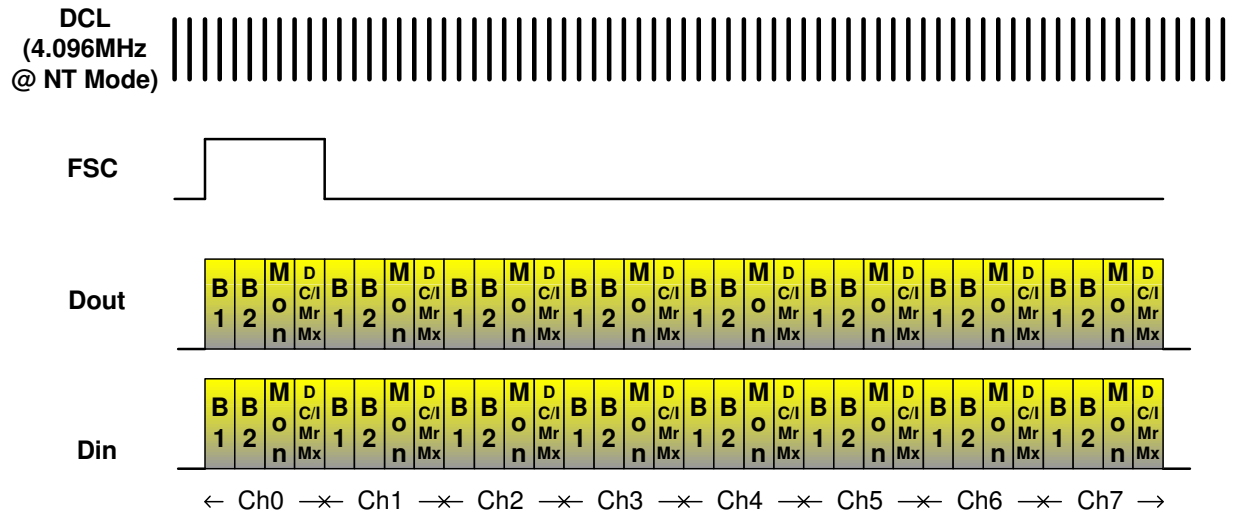


Figure 2: GCI simplified timing diagram for NT mode

NT Mode (f_{DCL} = 4096kHz, f_{Data} = 2048kHz)

3.1.3 PCM Highway mode

The PCM Highway mode of the UART interface provides a connection path between line transceivers (ISDN) and codecs. The M32C/83 set into PCM Highway mode can act like a switch backbone. In this mode 32 slots of 8bit width are available, bounded by the 8kHz FSC signal. Because the DCL frequency is equal to the data rate at the Data up/downstream lines, DCL clock will not divided, by the internal functionality of the UART.

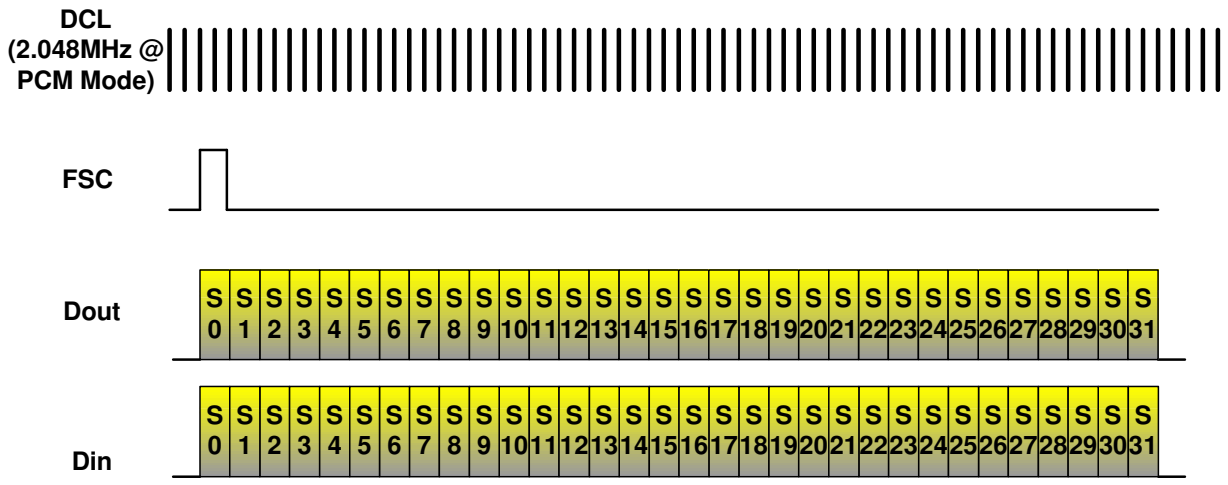


Figure 3: GCI simplified timing diagram for PCM highway mode

PCM Mode (f_{DCL} = 2048kHz, f_{Data} = 2048kHz)

3.2 GCI interface support of M32C/83

Because GCI intercommunication needs a pure serial interface, the UART2 of the M32C/83 supports the GCI functionality, by using an implemented hardware unit for GCI purpose. The GCI unit of the M32C/83 generates the Data Clock by dividing the f_{DCL} clock by two (except in PCM highway mode). Additionally the data scanning of the D_{IN} line or the data writing at D_{OUT} line can be synchronized to the rising edge of the FSC signal, using the synchronization function of the GCI unit.

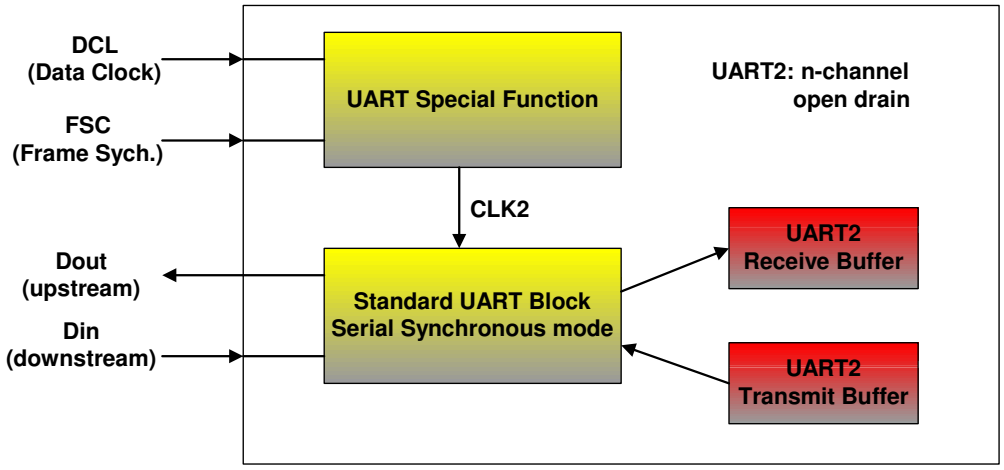


Figure 4: UART2 GCI interface schematic

Since Layer 1 is not implemented in the M32C/83, UART2 should be connected to a Layer1 device by following lines.

Function	M32C/83 (144pin package)			M32C/83 (100pin package)		
Dout	TxD2	Port7.0	Pin37	TxD2	Port7.0	Pin30
Din	RxD2	Port7.1	Pin36	RxD2	Port7.1	Pin29
DCL	CLK2	Port7.2	Pin35	CLK2	Port7.2	Pin28
FSC	CTS2	Port7.3	Pin34	CTS2	Port7.3	Pin27

Table 1: GCI related pins of M32C/83

Using the PCM Highways three different kinds of timings can be used.

- Referenced to the rising edge of DCL
- Referenced to the falling edge of DCL
- Referenced to the rising edge of DCL and FSC

To make clear the effect of the clock edge setting, please refer to figure 5. The M32C/83 satisfy with its simple and flexible timing opportunities, enabling the connectivity to most available codecs and line interfaces.

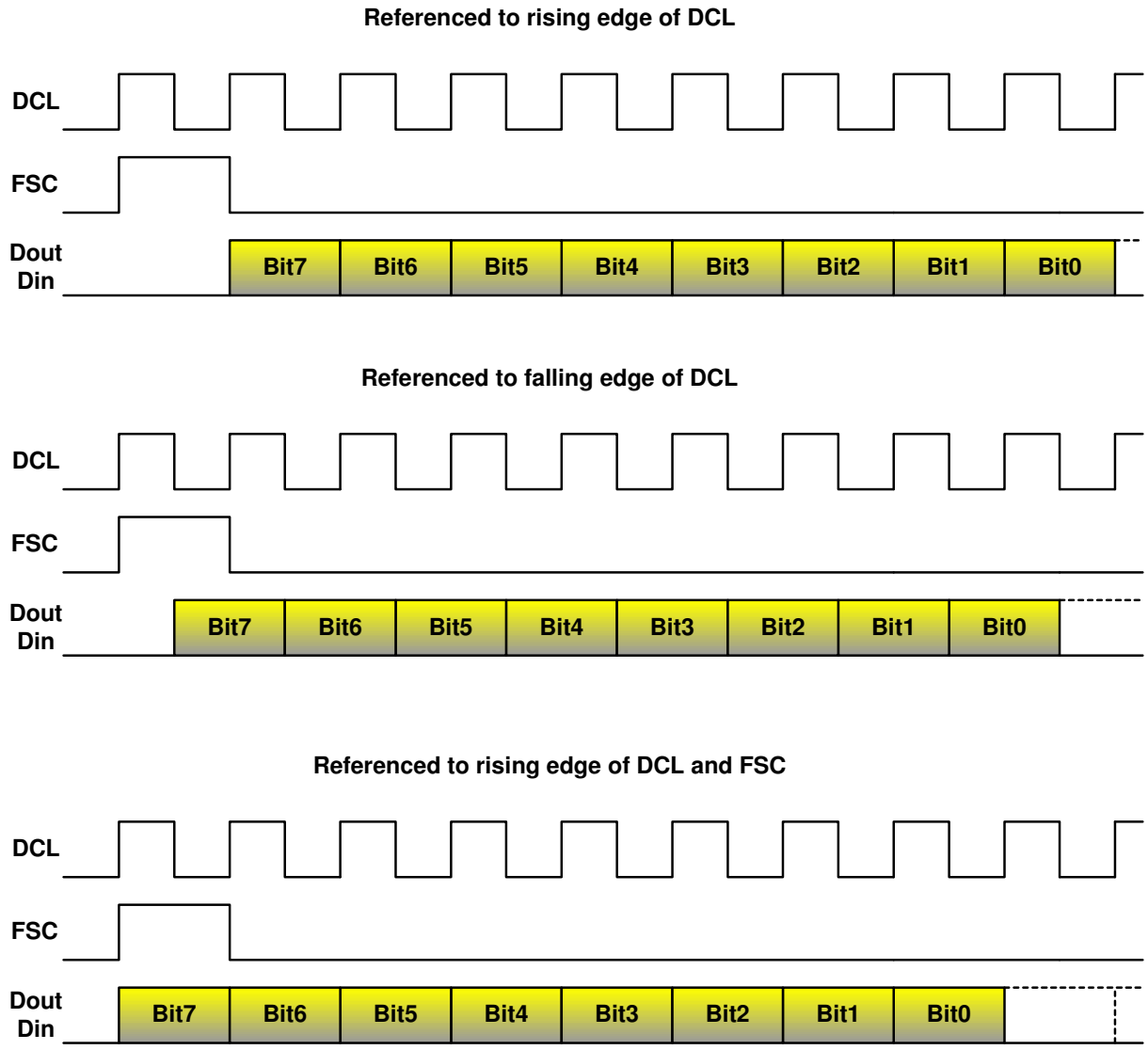


Figure 5: PCM Highway timing settings

To fulfill the timing for the synchronization at rising edge of FSC and DCL, an external circuit is required. This arrangement should generate a small delay (around some nsec) between rising edge of DCL and FSC. To do so, different approaches are possible to delay the DCL signal, e.g. RC filter or using some logical gates as delay line.

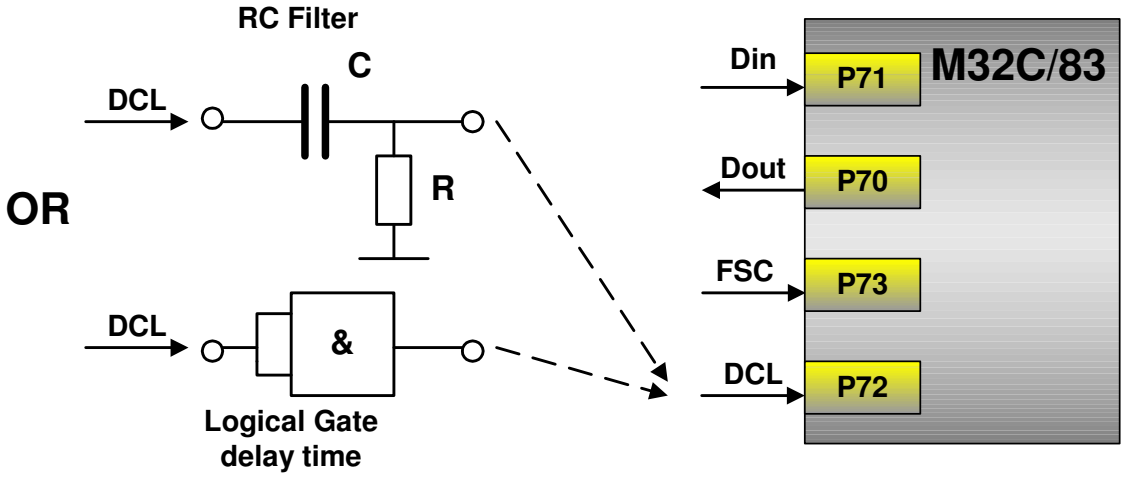


Figure 6: Possible delay lines for DCL signal, if referenced to the rising edge of DCL and FSC

3.3 GCI Software Driver

The general transfer of data from GCI interface to RAM buffer or from RAM buffer to GCI is handled by the DMA0 and DMA1 of the M32C/83. Thereby the DMA0 is initialized to transfer the received data automatically from receive output register RxD2 to a shadow buffer system. Moreover DMA1 is initialized to transfer the transmit data automatically from a shadow buffer system, to transfer input register TxD2.

After a complete frame has been received by DMA0, an interrupt will be generated, where the buffers of the shadow system for receive and transmit (no DMA1 interrupt is needed) will be swapped. The shadow buffer for each communication direction consists of one array (ucSWD_GCI_RcvBuffer, ucSWD_GCI_SndBuffer) and two pointers, whereby one of the pointer contains the address of the first element of the array and the other is points to the middle of this array.

The number of transferred bytes depends on the selected GCI mode support. In TE mode 12 bytes, in NT mode 32 bytes and in PCM Highway mode 32 bytes will be transferred by DMA, before an interrupt is entered, to do the necessary pointer swapping.

Additional to the pointer rearrangement inside the interrupt routine SWD_GCI_RcvSndRdy, a function call will be executed. The function call of this SWD_GCI_CallBackFunction, which is actually a pointer to a user defined function, so modification of the driver itself is not necessary, to extend the handling of the GCI data.

For this, a function is used as parameter for the SWD_GCI_Init() function call at the setup process, whereby the selected function should include the user GCI data routing/handling. Due to the logical link between FSC and DMA0 interrupt, the function will be entered every

125µsec. Therefore the function is called Service_8KHZ in this sample. Anyway, the user can use an own function instead, just by referring to the function at the GCI init process.
The SWD_GCI_CallBackFunction call includes two pointers to the GCI buffers as parameter, to allow access to the current up/down stream data.

Inside this routine the user can do all necessary reading, writing, modification or routing of the different data bytes of the CGI data lines, by referring to the pointer plus the selected slot number.

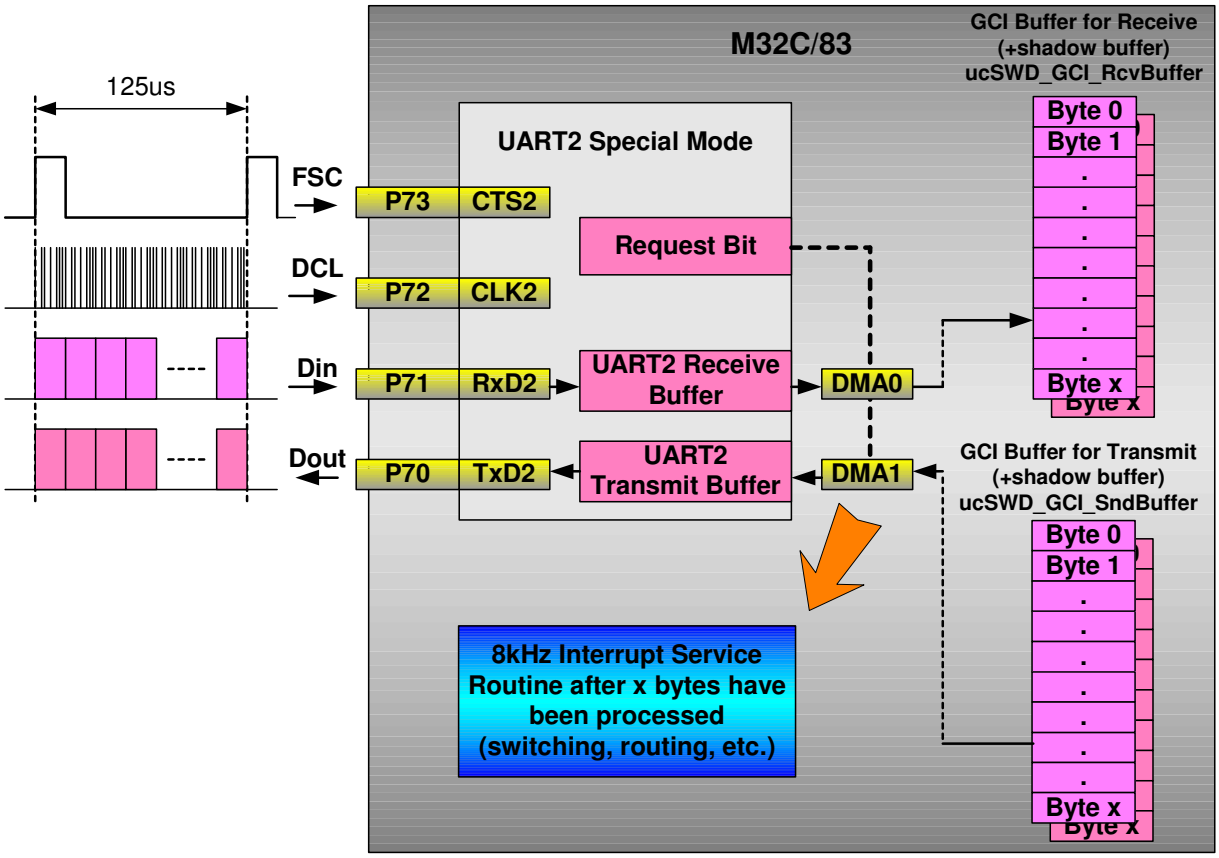


Figure 7: GCI driver concept

3.4 GCI Software Driver Flow diagram

The described Software Driver for GCI purpose is written in C-Source.
The GCI driver consists of SWD_GCI.c and SWD_GCI.h file. In the user code at least the SWD_GCI_Init() and SWD_GCI_Start() functions have to be called. Additional a Service_8KHZ function have to be implemented in the user code, which is the parameter of the SWD_GCI_Init() call as well. The GCI mode can be selected in the SWD_GCI.c local

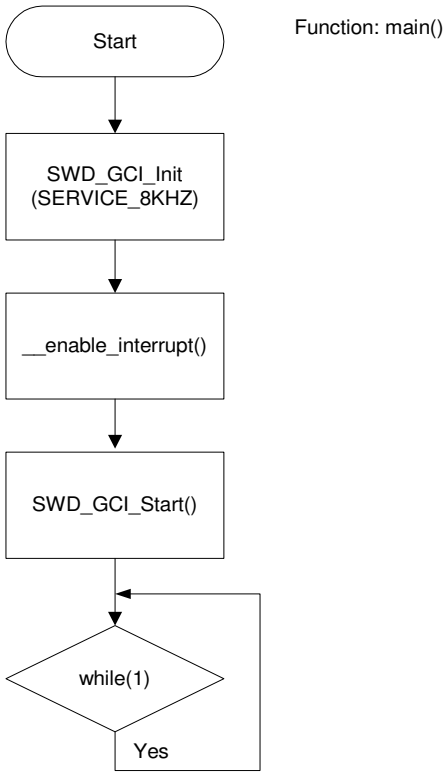
header part. Furthermore the synchronous behavior of the data lines signals regarding the DCL signal can be selected there as well.

The following function will be used:

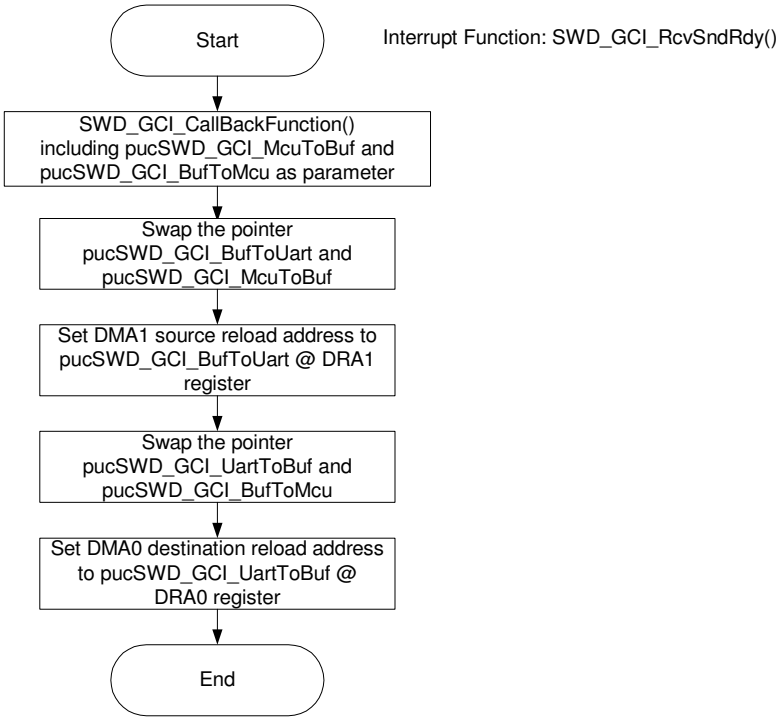
Functionname	Purpose
Main()	Main routine
SWD_GCI_RcvSndRdy()	DMA0 interrupt service routine
Service_8KHZ()	Interrupt Callback function of user
SWD_GCI_Init().c	Initialization of GCI interface general
SWD_GCI_SetRcv()	Initialization of GCI interface receive part (DMA0)
SWD_GCI_SetSnd()	Initialization of GCI interface transmit part (DMA1)
SWD_GCI_Start()	Start of GCI function
SWD_GCI_Stop()	Stop of GCI function

Table 2: Functions of the GCI software driver

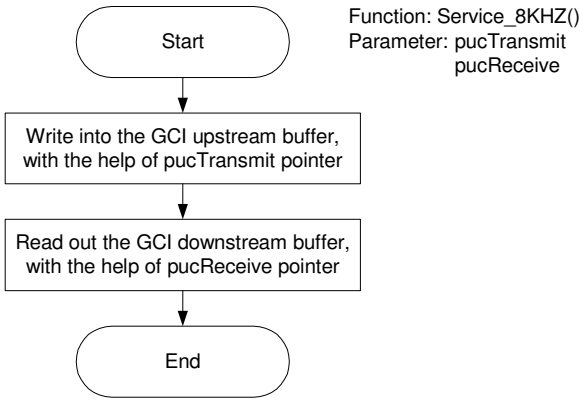
3.4.1 Main() function



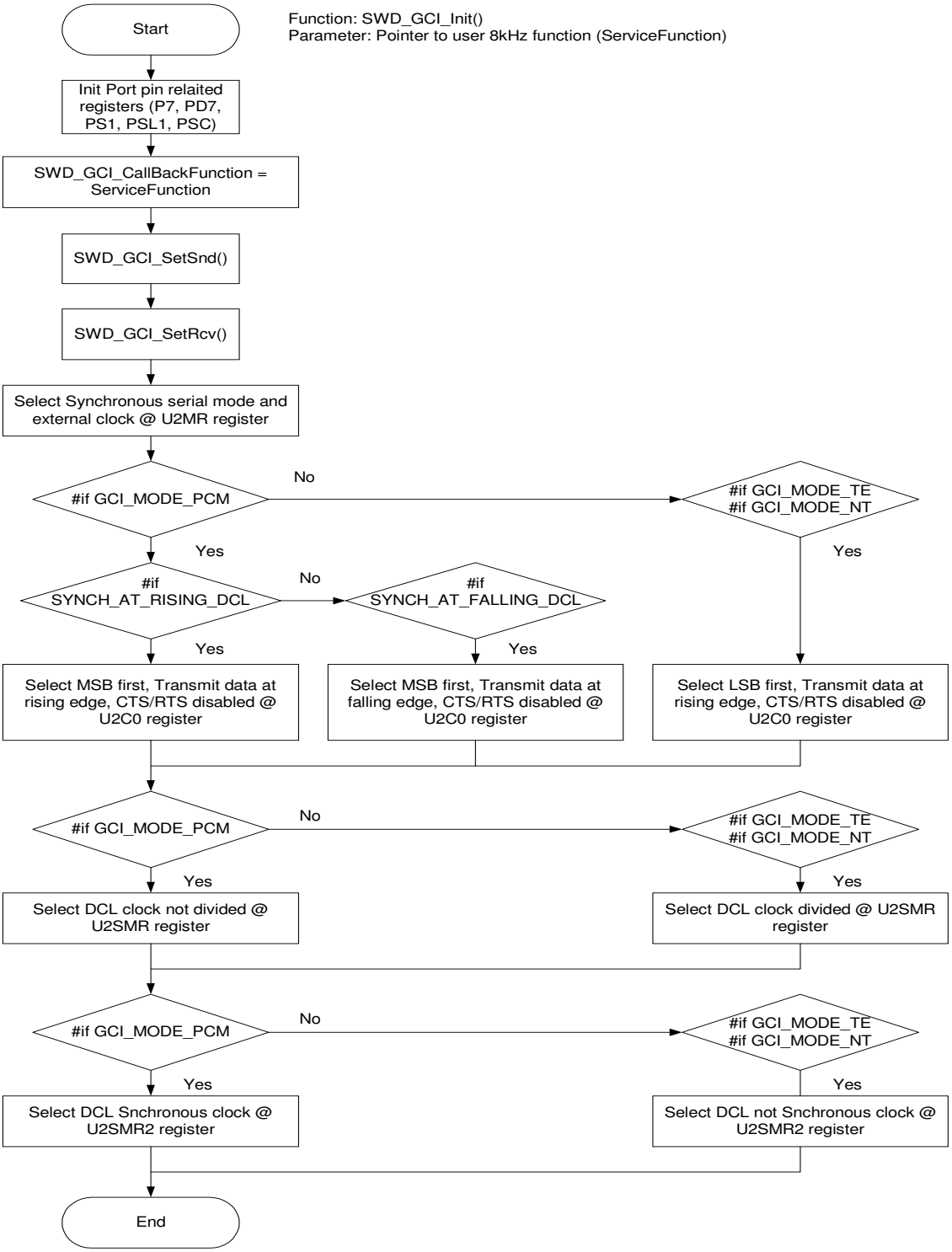
3.4.2 SWD_GCI_RcvSndRdy() function



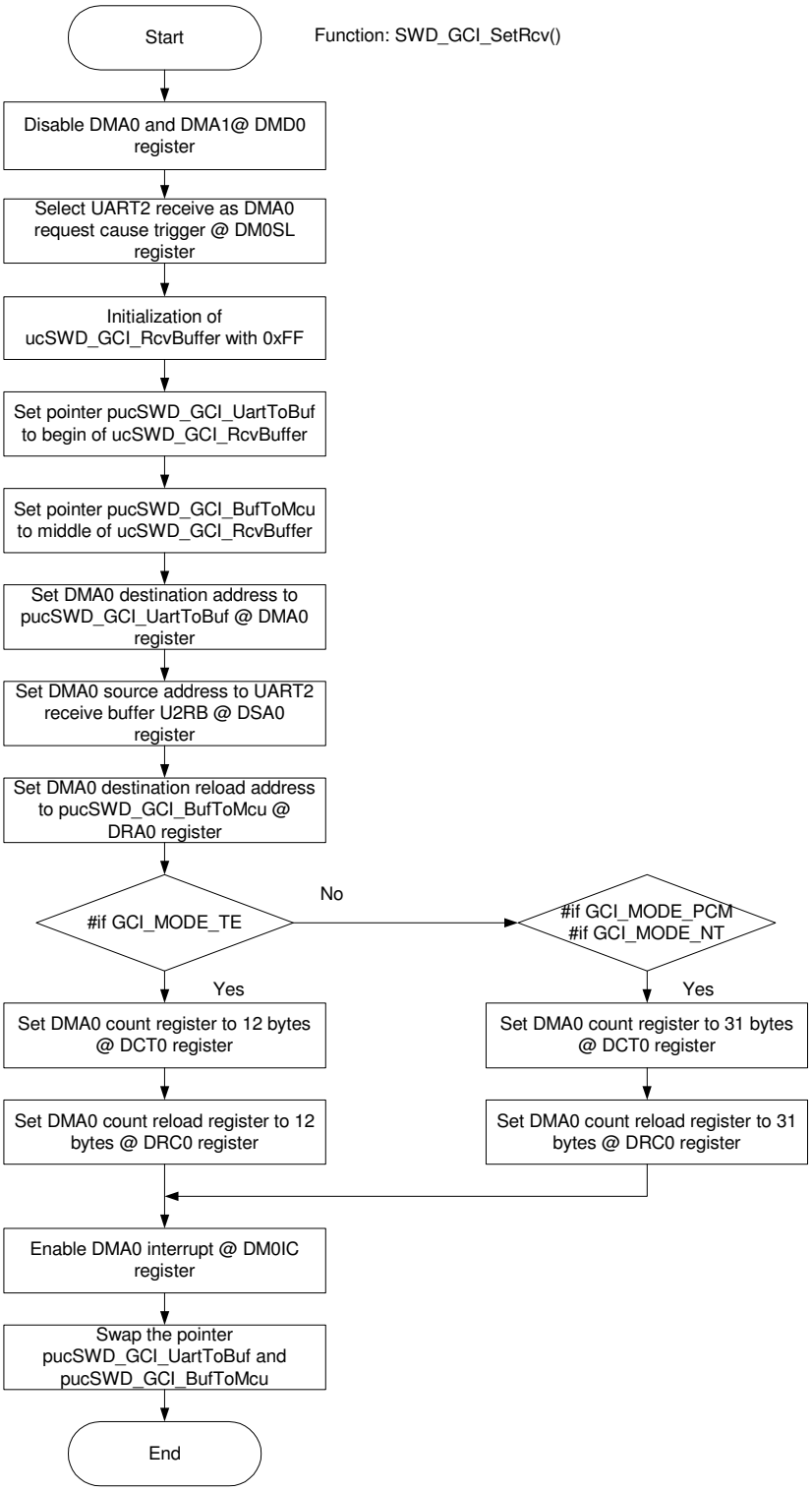
3.4.3 Service_8KHZ() function



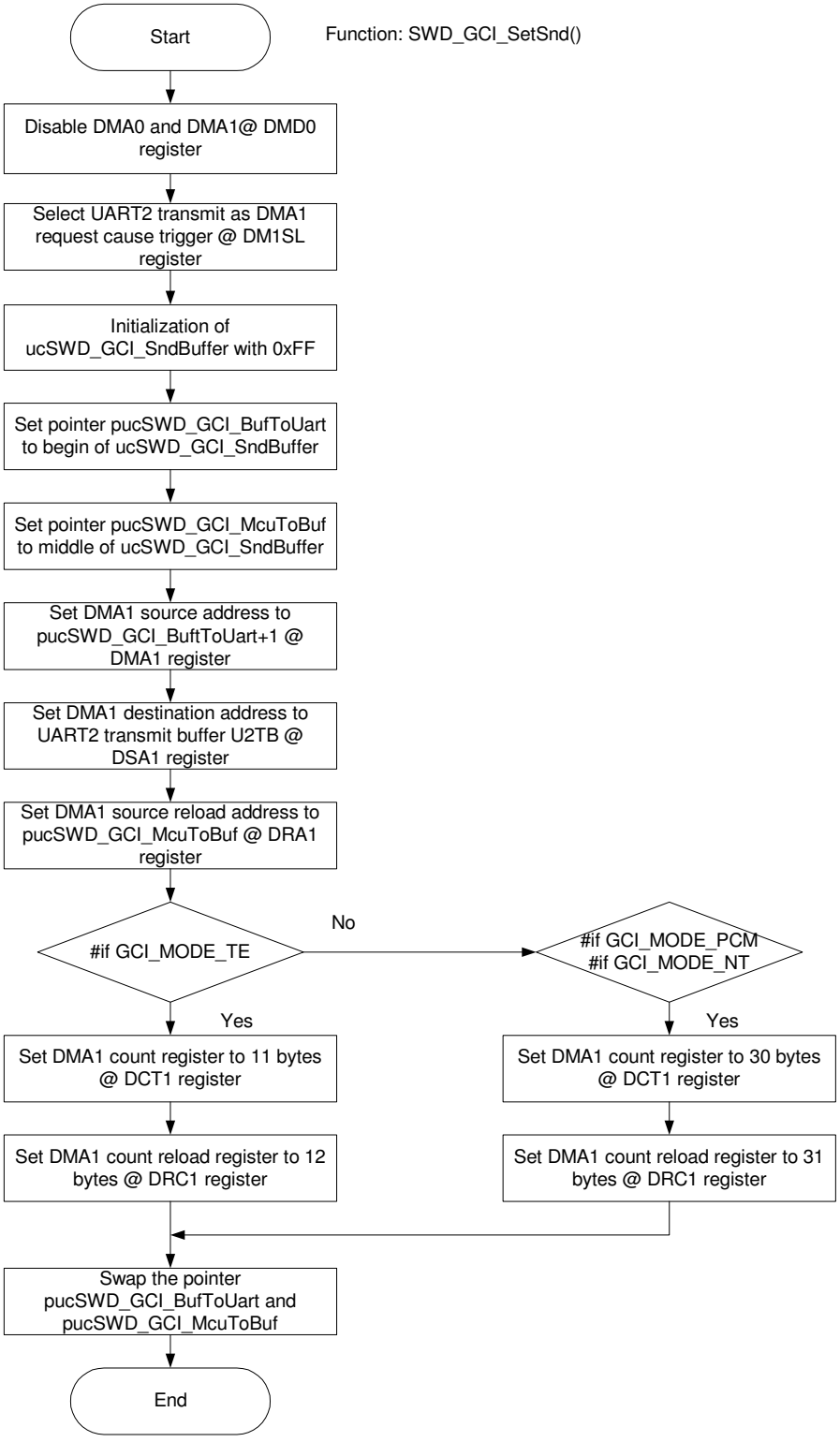
3.4.4 SWD_GCI_Init() function



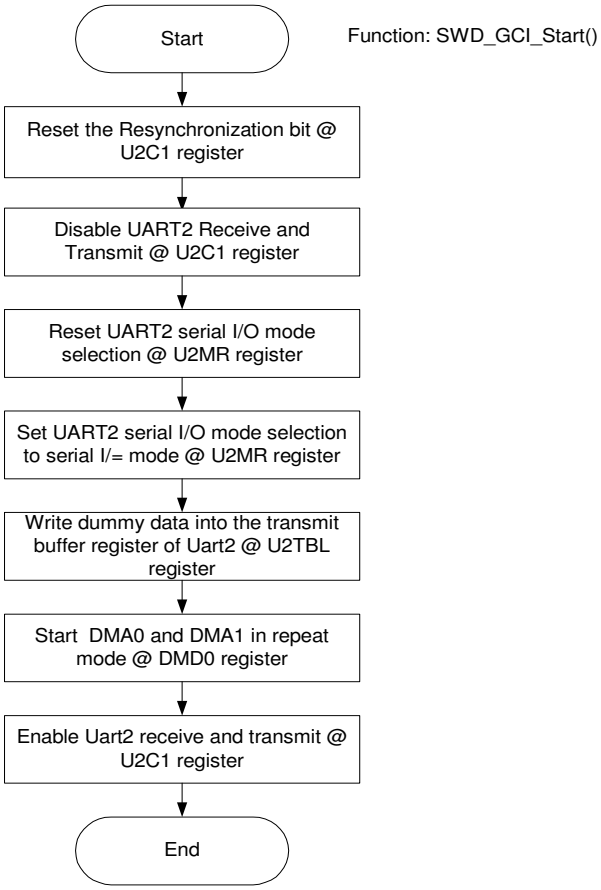
3.4.5 SWD_GCI_SetRcv() function



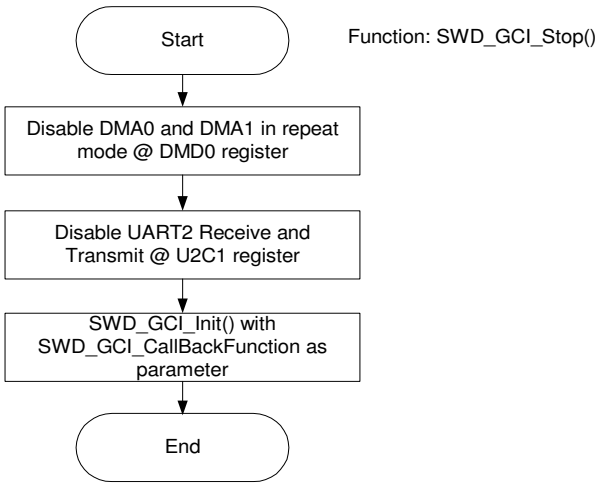
3.4.6 SWD_GCI_SetSnd() function



3.4.7 SWD_GCI_Start() function



3.4.8 SWD_GCI_Stop() function



4 HDLC feature

4.1 General Description of HDLC

HDLC is a universal standard error detecting protocol, which allows code transparent binary transmissions of data. The job of the HDLC layer is to ensure that data, passed up to the next layer, has been received exactly as transmitted (i.e. error free, without loss and in the correct order). Another important job is flow control, which ensures that data is transmitted only as fast as the receiver can receive it. The general purpose of this frame construct is to carry the Layer 3 information.
A HDLC frame consists of different blocks.

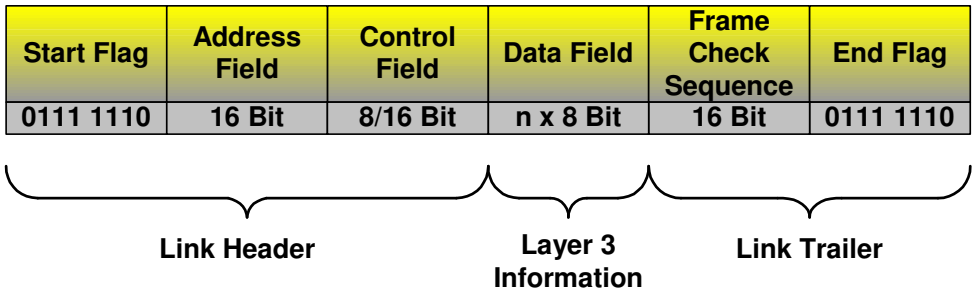


Figure 8: Outline of HDLC frame

The beginning and end of an HDLC frame are marked by flag bytes "01111110" binary. No flag character may appear within the frame. To enforce this requirement, the data may need to be modified in a transparent manner. Therefore a binary 0 is inserted after every sequence of five 1s binary by the transmitter, this is called bit stuffing. Thus, the longest sequence of 1s of the link that may appear is "0111110", one less than the flag character.
The receiver, upon seeing five 1s, examines the next bit. If this bit is 0, the bit is discarded and the frame continues. If it is 1, this must be the flag sequence at the end of the frame.
At the end of the frame, a Frame Check Sequence (FCS) is used to verify the data integrity. The FCS is a CRC calculated using polynomial $x^{16}+x^{12}+x^5+1$.
Between HDLC frames, the link idles. Most synchronous links constantly transmit data; these links can transmit all 1s during the inter-frame period (called mark idle), or all flag characters (called flag idle).
Usually when referring to HDLC in ISDN areas, people mean LAPD a deviate of HDLC protocol. LAPD is a slightly modified version of HDLC.

4.2 Intelligent I/O Group of M32C/83

For HDLC functionality the M32C/83 use two hardware implemented Intelligent I/O Groups. Each of these two blocks provides a full-duplex HDLC channel includes following hardware to realized HDLC functionality:

- Zero-bit-deletion/insertion
- CRC check according to CRC-CCITT
- Start/end flag detection
- Abort flag detection

This HDLC is a pure physical interface. It does not support any high-level layer functions.
The following block diagrams summarize the HDLC related portion from Intelligent I/O Group 0. The hardware realization for the Intelligent I/O Group 1 is transparent.
Each of these Intelligent I/O Groups can be segmented into following three blocks:

- Basetimer Clock generation
- Receive HDLC unit
- Transmit HDLC unit

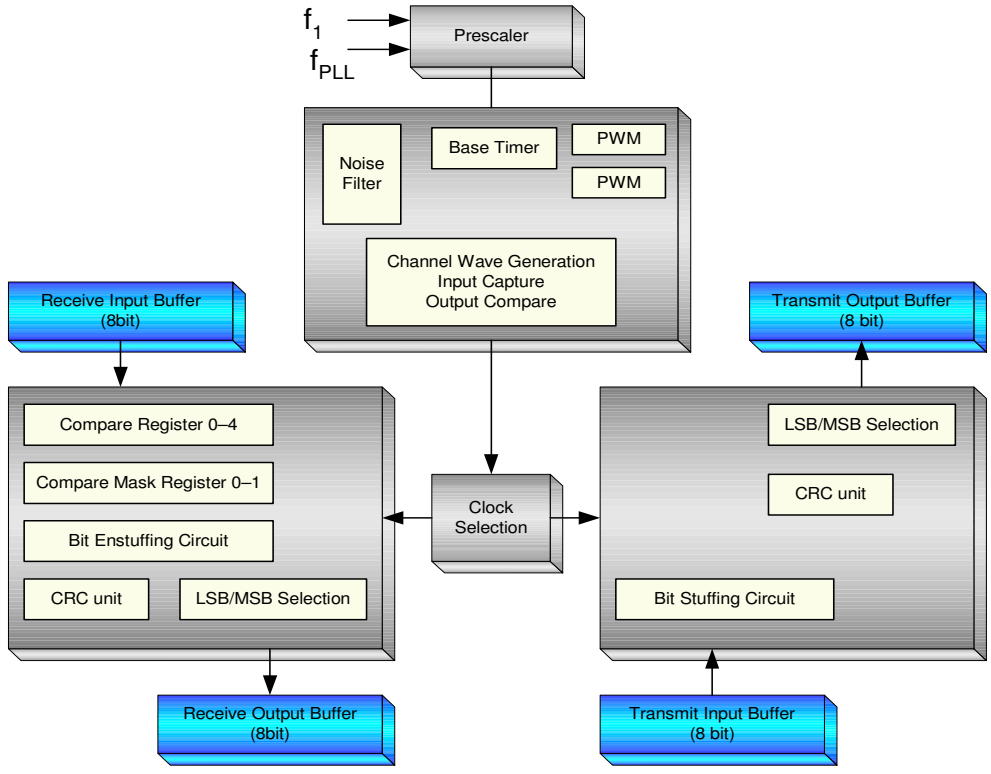


Figure 9: Outline of Intelligent I/O Group 0

In the following documentation, within some register names may "i" appear. This is a variable for Intelligent I/O Group 0 and 1, because both Groups have similar registers and therefore this documentation is transparent for both.

4.2.1 Basetimer clock generation

The clock for receive and transmit part is generated by the freerunning Basetimer, plus the usage of two additional compare registers. Each unit, receive and transmit has their own compare register.

- GiPO0 is the compare register for the Receive unit
- GiPO1 is the compare register for Transmit unit.

Is the value of the Basetimer equal to a compare register value, a logical high output will be generated by the compare register. A rectangle clock is generated by this logic output and feed to the appurtenant circuit. The following diagram should explain this function:

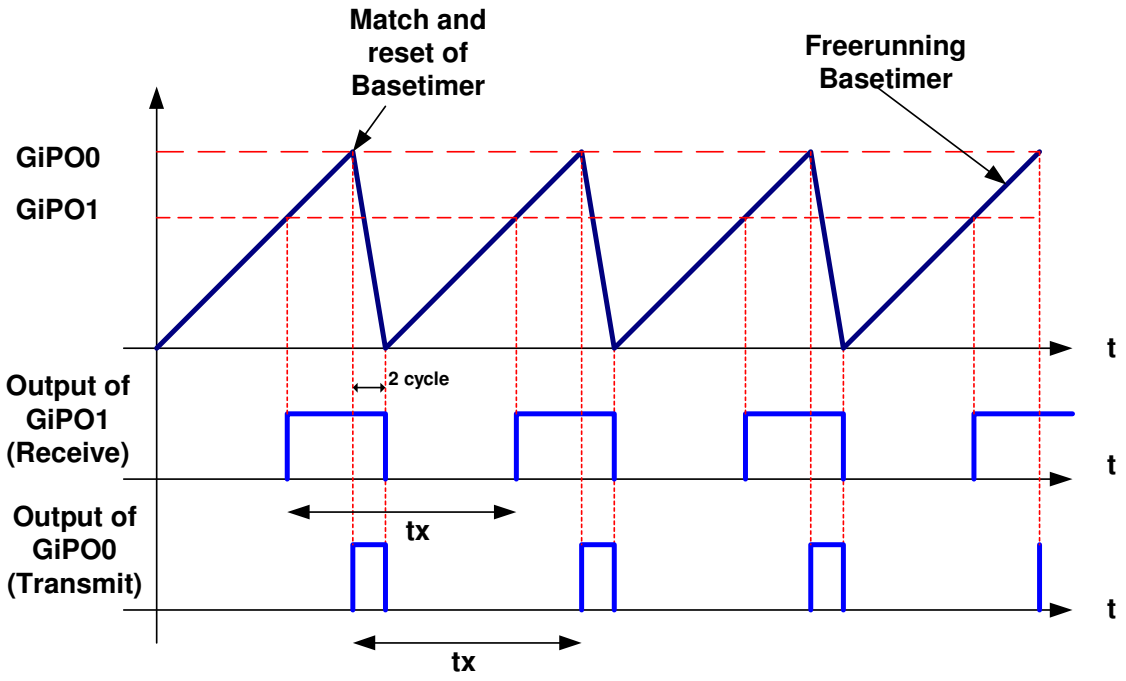


Figure 10: Handling of Basetimer and Compare registers

Is the value of the Basetimer equal to the GiPO0 register value, the Basetimer will be reset automatically (takes two cycles) and the clocks for receive and transmit block return into logical low output state. Please make sure that the value of GiPO0 register is higher than the

GiPO1 one. The shapes of the clocking signals are different from each other, but this doesn't matter, because only the rising edge is important for the following units and this frequency is same for receive and transmit side. In this driver, basetimer frequency is set to around 1.6MHz whereby f_{xin} is 20MHz.

4.2.2 Receive HDLC unit

The receive unit is supplied with a clock which is generated by the output of the compare register GiPO1.
The incoming HDLC data, which have been removed from GCI frame, should be written into the receive input buffer (GiRI). From the receive buffer the data will be serial clocked to the compare shift register (GiDR) and to the bit enstuffing unit.
The compare unit does a permanent comparison between current serial data stream and the values inside the data compare registers. For the registers GiCMP0 and GiCMP1 an additional mask register called GiMSK0 and GiMSK1 is available. If the current data matches one of this compare registers, a trigger signal CMP0T-CMP3T will be released. This signal can be used in the driver software for flag recognition, e.g. start/end flag or abort flag. Therefore the recommended setting for the compare registers is:

- GiCMP0 = 0xFF and GiMSK0 = 0x7F used for abort detection
- GiCMP3 = 0x7E used for start/end flag detection

The other compare and mask register will not be used for standard HDLC processing.

The bit enstuffing unit scans the incoming serial data stream, for a sequence of five "1" binary. If the next bit of such sequence is "0", the bit is discarded and the frame continues. Meanwhile, the enstuffing unit sends a stop signal to the clock wait unit. So, the deleted "0" bit is not shifted anymore to the output buffer and CRC generation unit. Shortly after this deletion the stop signal is withdrawn, so standard clocking take place again. If the bit after a sequence of five "1s" is also "1" a start/end flag is been detected by the compare unit and the responsible compare register outputs a trigger signal.
The enstuffed data stream is clocked to the CRC controller and to the receive shift register. If the receive shift register is full, the data will be backup into the receive output register, where the data can be read out by the software driver. During this, the CRC unit generates permanent the CRC of the incoming data. If an end flag is detected by the compare unit, the CRC will be moved into the receive CRC register, where the CRC can be read out. Due to these circumstances the CRC generation stops working after end flag is detected, so the complete end flag is also involved in the current generated CRC code. Because of this

algorithm the generated CRC is for every received frame equal. That means don't care what kind of frame was received, the CRC is 0x0B9F hex, in case the frame was received without error.

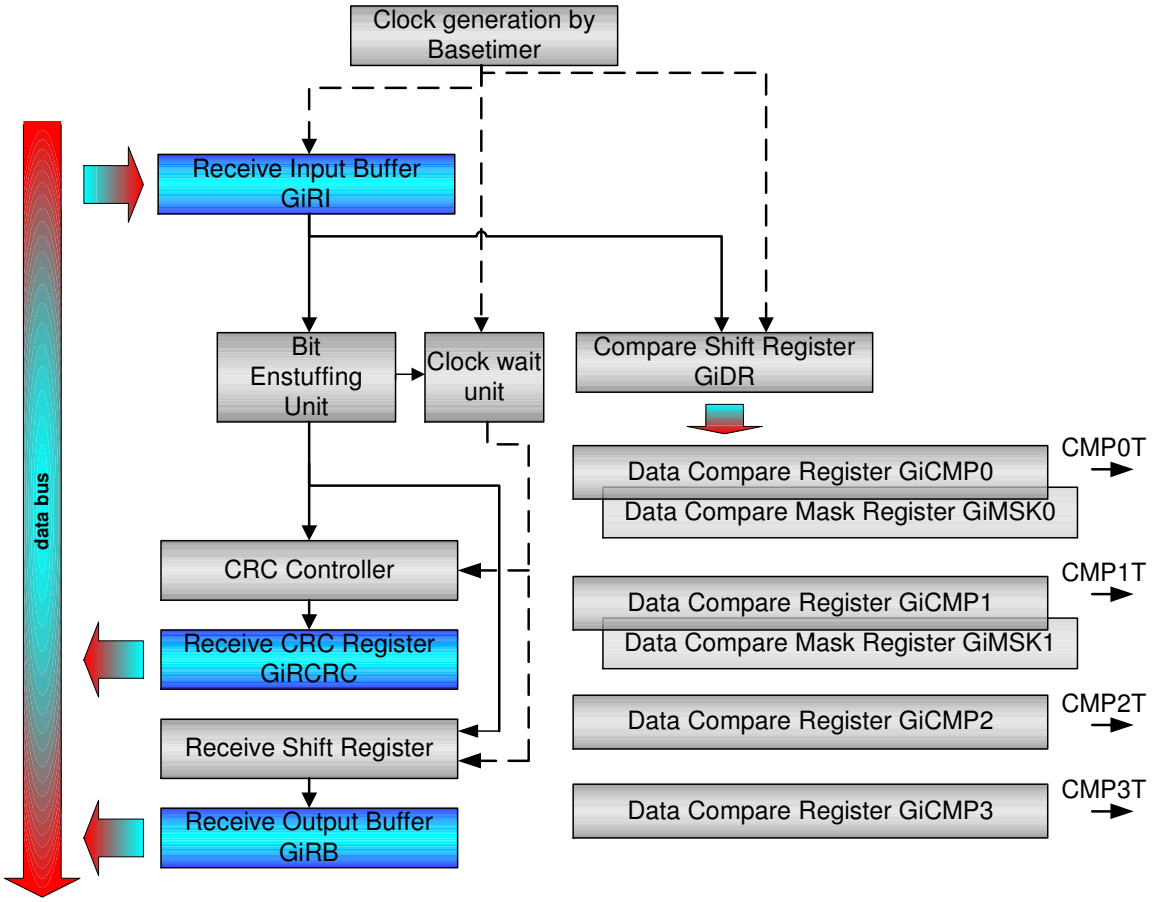


Figure 11: Outline of Receive HDLC unit

4.2.3 Transmit HDLC unit

The transmit unit is supplied with a clock which is generated by the output of the compare register GiPO0.

To transmit data using the HDLC block, the data have to be written into the transmit buffer (GiTB). First of all a start flag should be written into this register, followed by the rest of the HDLC frame. For transmission of the start flag, the bit stuffing unit and CRC unit should be disabled. The data will be shifted, with help of transmit shift register to the bit stuffing unit. The bit stuffing unit scans the incoming serial data stream, for a sequence of five "1s" binary. If such data stream is detected, the bit stuffing unit stops clocking for the transmit shift register

and CRC controller, by using the clock wait unit. Therefore the bit stuffing unit is now able to insert a zero bit into the data stream. Shortly after this insertion the stop signal is withdrawn, so standard clocking take place again and the frame continues. The CRC unit generates permanent the CRC of the incoming unstuffed data stream. Is the HDLC frame is completed the CRC can be read out of the transmit CRC register (GiTCRC). Because the CRC itself has to be stuffed, too. The CRC code has to be written into the transmit buffer. The stuffing unit does the stuffing of the CRC, like it has done for the previous data and the result will be available in the transmit output register (GiTO). To finalize the HDLC frame an end flag should be attached by the transmit unit, whereby the stuffing unit and CRC controller are disabled again.

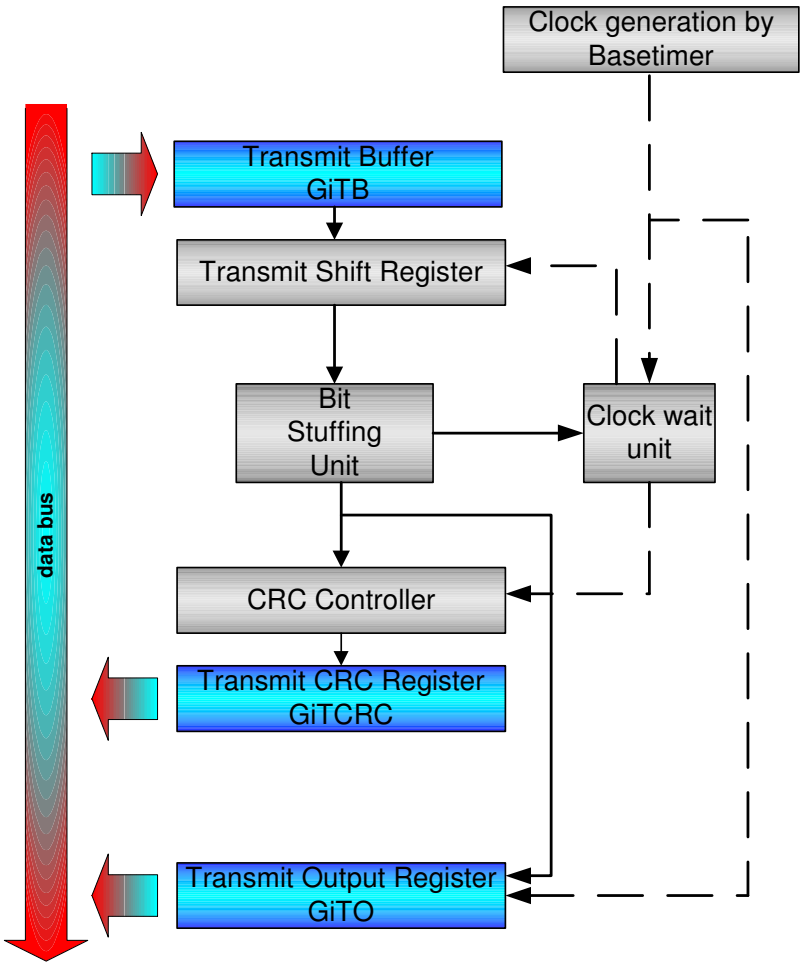


Figure 12: Outline of Transmit HDLC unit

4.3 HDLC Software Driver

This Software Driver supports two HDLC channels (HDLC0 and HDLC1 block of Intelligent I/O group 0 and 1), depending on define pre-processor directive in the main.c file. The needed initialization of the HDLC blocks will be done by execution of the SWD_HDLC0_Init() and/or SWD_HDLC1_Init() function. Because the HDLC driver itself needs an environment for operation, it is linked to the GCI driver to show the functionality.

In the Service_8KHZ() routine, a specific slot of the GCI downstream frame will be extracted and transferred to the HDLC receive input register, processed and finally read out of the receive output register of the HDLC block. For GCI upstream direction, user data will be input to the transmit input buffer of the HDLC block, processed and read out of the HDLC transmit output buffer register and inserted into the specific slot of the GCI upstream frame.

To generated this necessary routing following order of function calls should be followed in the Service_8KHZ callback function:

1. SWD_HDLC0_RcvOut
2. SWD_HDLC0_RcvPoll
3. SWD_HDLC0_RcvIn
4. SWD_HDLC0_SndPoll
5. SWD_HDLC0_SndIn
6. SWD_HDLC0_SndOut

This is the order for the function calls for HDLC0 generation only. If HDLC1 is needed as well please use the similar functions for the HDLC1 block afterwards. Furthermore, following two paragraphs describe the HDLC0 block, but HDLC1 block, function and handling is similar.

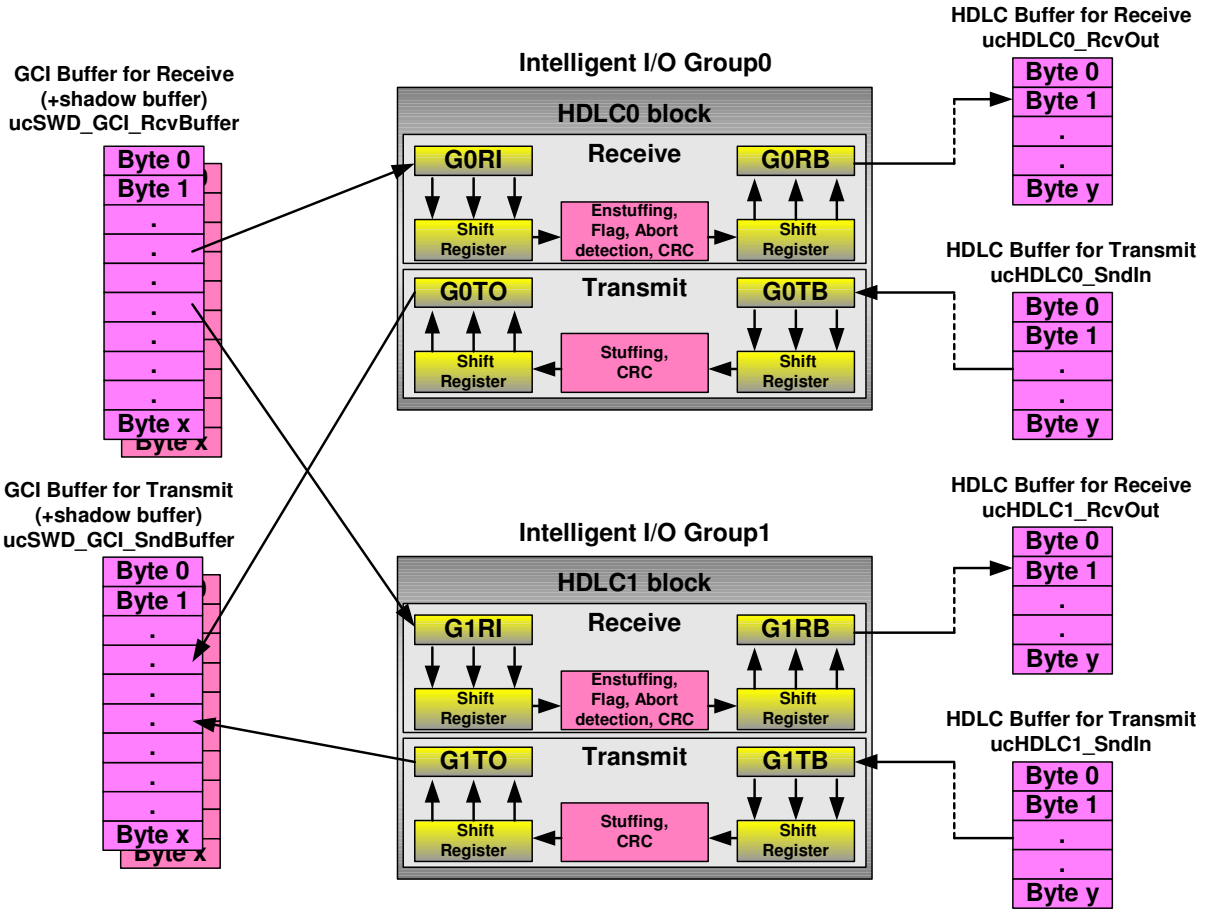


Figure 13: HDLC driver concept

4.3.1 Receive HDLC

The first step in the HDLC driver is to check, whether data is available in receive output buffer G0RB. Therefore the function SWD_HDLC0_RcvOut() will executed and the data of the output buffer register will be stored in the buffer array ucHDLC0_RcvOut. Additional the index counter ulSWD_HDLC0_RcvIndex for the array access will be incremented. If this counter exceeds the limit, it will be set to a fix value and an overflow counter will be incremented.
The next step is calling of SWD_HDLC0_RcvPoll() function. Within this routine the handling of the compare registers of the HDLC receive block is done. If no interrupt request flag have been set by the compare registers, the routine will be left without action, but if a start/end or abort detection flag is set, a dummy read is done at the receive output register, the flag itself and receive output buffer full interrupt request flag will be reset.

In case start/end flag has been detected, abort detection will be enabled. The current amount of received bytes will be compared with a minimum frame length parameter to determine, whether flag is start or end flag. If the current frame length is smaller than the minimum length value, the flag will be identified as start flag. If a start flag has been detected already in one of the previous passes, the flag is identified as end flag. Now the CRC will be read out of G0RCRC register and will be compared to the expected CRC.
In case an abort flag is detected, abort flag detection will be disabled until next start/end flag will be detected and a possible previous recognized start flag and current frame as well would be discarded.
Then the selected byte of the downstream GCI frame will be transferred to the SWD_HDLC0_RcvIn() function as parameter, containing pointer to storage location of GCI frame and a offset for the selected byte. Within SWD_HDLC0_RcvIn() function the received byte is written into the receive input buffer of the HDLC0 block.

4.3.2 Transmit HDLC

First action for the transmit HDLC block treatment is to execute the SWD_HDLC0_SndPoll() function. If a transmission has been started already with the SWD_HDLC0_SndIn() function, a state machine procedure will start, to do the necessary settings for bit stuffing, CRC generation and flag transmission, otherwise the function will be left without action.
The state machine consist of following states, which will be executed in the following order as well:

State	Purpose
SWD_HDLC_SND_STATE_DATA	Transmission of data and enabling of bit stuffing
SWD_HDLC_SND_STATE_CRCL	Transmission of low byte of CRC
SWD_HDLC_SND_STATE_CRCH	Transmission of high byte of CRC
SWD_HDLC_SND_STATE_FLAG	Disable of bit stuffing and transmission of end flag
SWD_HDLC_SND_STATE_FILL	Transmission of fill byte
SWD_HDLC_SND_STATE_END	Final state

Table 3: States of SWD_HDLC0_SndPoll() function

In the first state the bit stuffing unit will be enabled and the data array, where pucSWD_HDLC0_SndInput is pointing to, is transferred to the G0TB register. Additional this data is input to the standard CRC circuit for transmit CRC generation. If the frame is complete transferred to G0TB register, the generated CRC will be read out in the next state and also transferred to the G0TB register, to process the bit stuffing. Afterwards the bit stuffing unit will be disabled and an end flag is transferred to the G0TB register, followed by filling data to

assure that the end flag is completely clocked through the transmit HDLC block. Finally, the final state disables the transmit block and reset all possible open requests as well as some dummy read of G0TO.

Please note, that the usage of transmit HDLC block internal CRC generation circuit is not recommended, if no interrupt routine is used for HDLC handling. Therefore, this driver utilized the standard CRC generation circuit of the M32C/83 for transmit CRC generation.

The SWD_HDLC0_SndIn() function call, includes a pointer and a length variable as parameter for indication of next transmit HDLC frame. As long as the return value of this function is ERROR, you should not change the selected HDLC frame. If the return value is OK the selected frame is already on transmission and a new frame can be selected for next transmission procedure. The transmit HDLC block is enabled, the transmit state machine is set to SWD_HDLC_SND_STATE_DATA and index variables are set to zero. Then the routine writes the start flag into the G0TB register, which actually starts the complete transmission. Then the address of the selected byte for the upstream GCI frame will be transferred to the SWD_HDLC0_SndOut() function as parameter, containing pointer to storage location of GCI frame and a offset for the selected byte. Within SWD_HDLC0_SndOut() function a byte will be written into the specific slot of the GCI frame. In case no byte is available at the transmit output buffer register G0TO, the software supposes that currently no HDLC frame has to be transmitted and instead of that a SWD_HDLC0_PAUSE byte is inserted to the GCI frame, which could be mark idle or flag idle data bytes. This can be selected by the user, via pre-processor directive in the local header part of the SWD_HDLC0.c and SWD_HDLC1.c file.

4.4 HDLC Software Driver Flow diagram

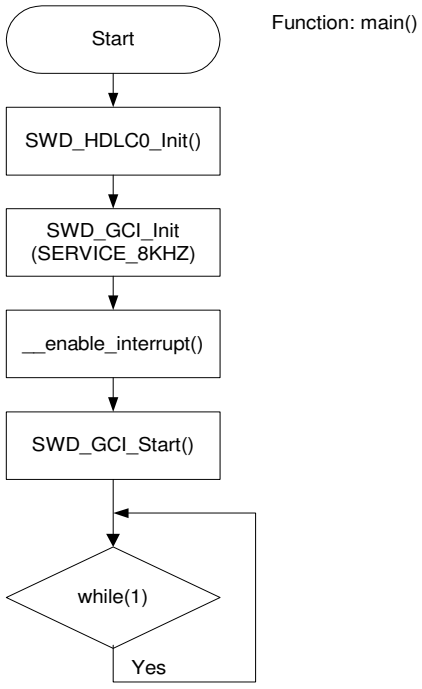
The described Software Driver for HDLC purpose is written in C-Source.
The HDLC driver consists of SWD_HDLC0.c, SWD_HDLC1.c, SWD_HDLC0.h and SWD_HDLC1.h file. In the user code the SWD_HDLC0_Init() and SWD_HDLC1_Init() functions have to be called. Additional, some action have to been done in a polling style. To combine GCI and HDLC functionality, it is recommended to use the Service_8KHZ function to handle these items.
The driver handles Intelligent I/O Group 0 and 1, anyway the flow diagram just for the HDLC0 group usage are attached, because flow diagram for HDLC1 is quite similar.

The following function will be used:

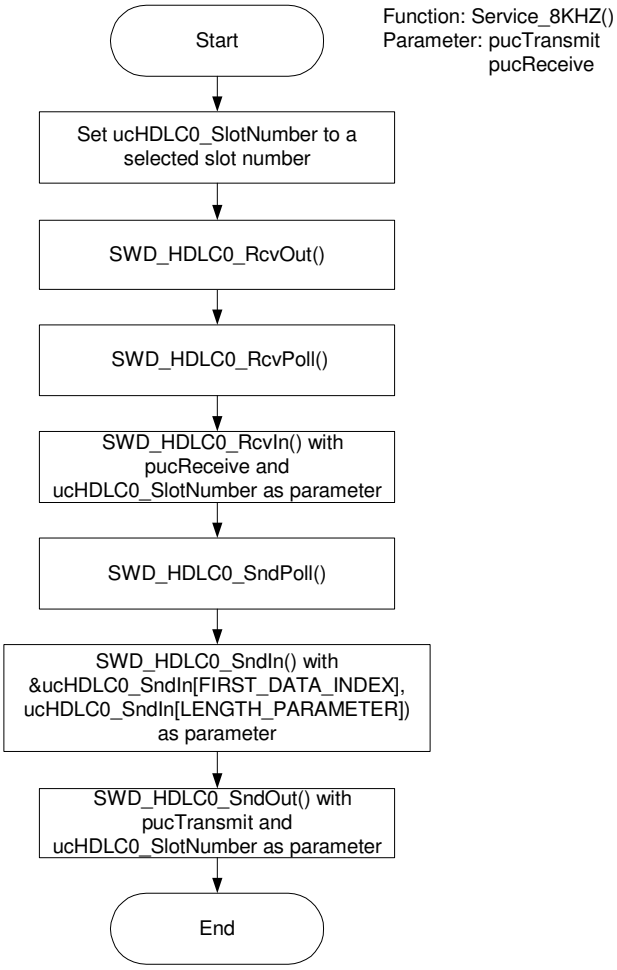
Functionname	Purpose
Main()	Main routine
Service_8KHZ()	Interrupt Callback function of user
SWD_HDLC0_Basetimer_Init()	Initialization of HDLC0 basetimer
SWD_HDLC0_Init()	Initialization of HDLC0 block in general
SWD_HDLC0_RcvIn()	Handling of data input for receive HDLC0 block
SWD_HDLC0_RcvOut()	Handling of data output for receive HDLC0 block
SWD_HDLC0_RcvPoll()	HDLC0 polling routine for receive communication
SWD_HDLC0_SndIn()	HDLC0 transmit start function
SWD_HDLC0_SndOut()	Handling of data output for transmit HDLC0 block
SWD_HDLC0_SndPoll()	HDLC0 polling routine for transmit communication
SWD_HDLC1_Basetimer_Init()	Initialization of HDLC1 basetimer
SWD_HDLC1_Init()	Initialization of HDLC1 block in general
SWD_HDLC1_RcvIn()	Handling of data input for receive HDLC1 block
SWD_HDLC1_RcvOut()	Handling of data output for receive HDLC1 block
SWD_HDLC1_RcvPoll()	HDLC1 polling routine for receive communication
SWD_HDLC1_SndIn()	HDLC1 transmit start function
SWD_HDLC1_SndOut()	Handling of data output for transmit HDLC1 block
SWD_HDLC1_SndPoll()	HDLC1 polling routine for transmit communication

Table 4: Functions of the HDLC software driver

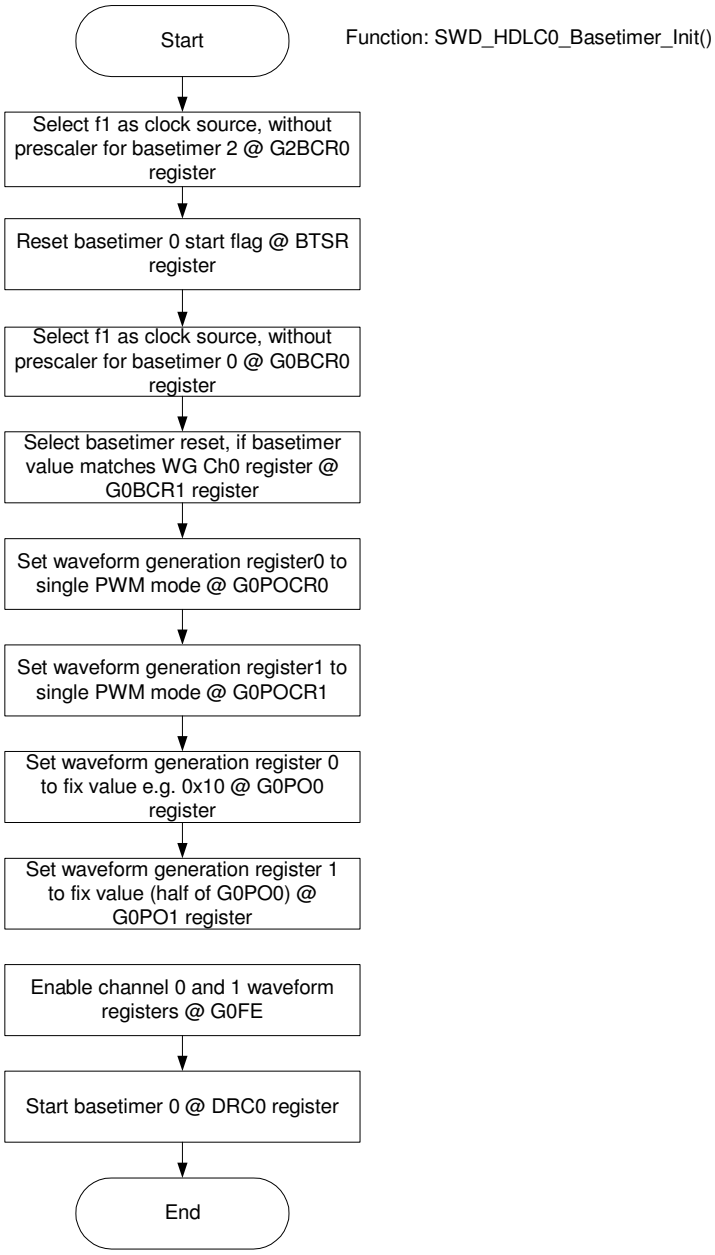
4.4.1 Main() function



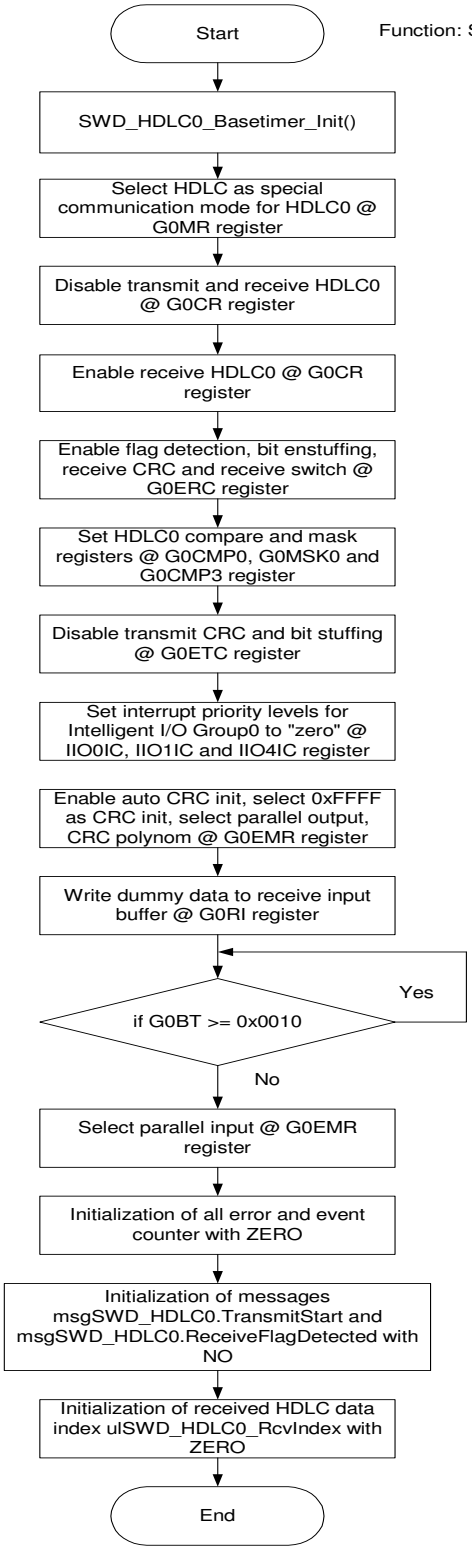
4.4.2 Service_8KHZ() function



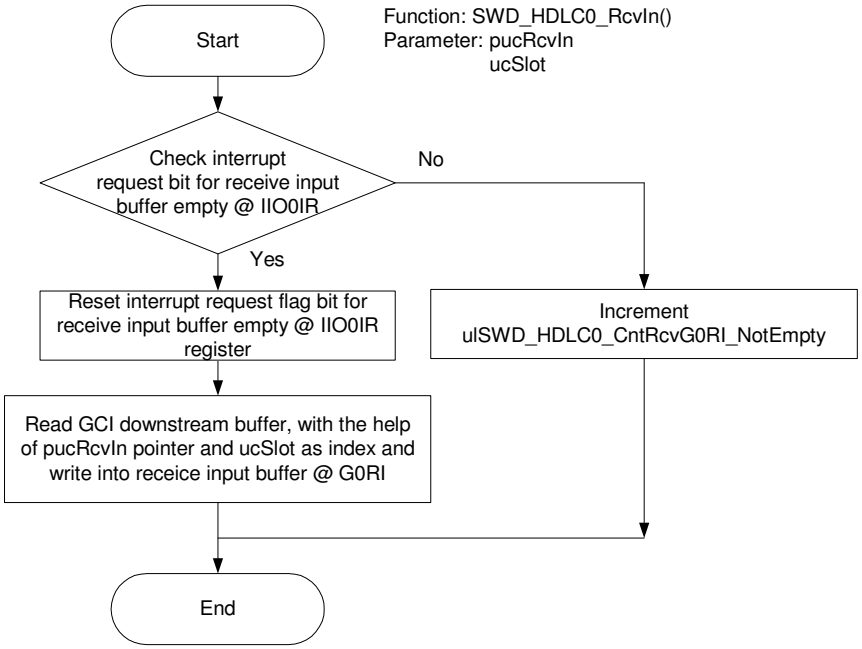
4.4.3 SWD_HDLC0_Basetimer_Init() function



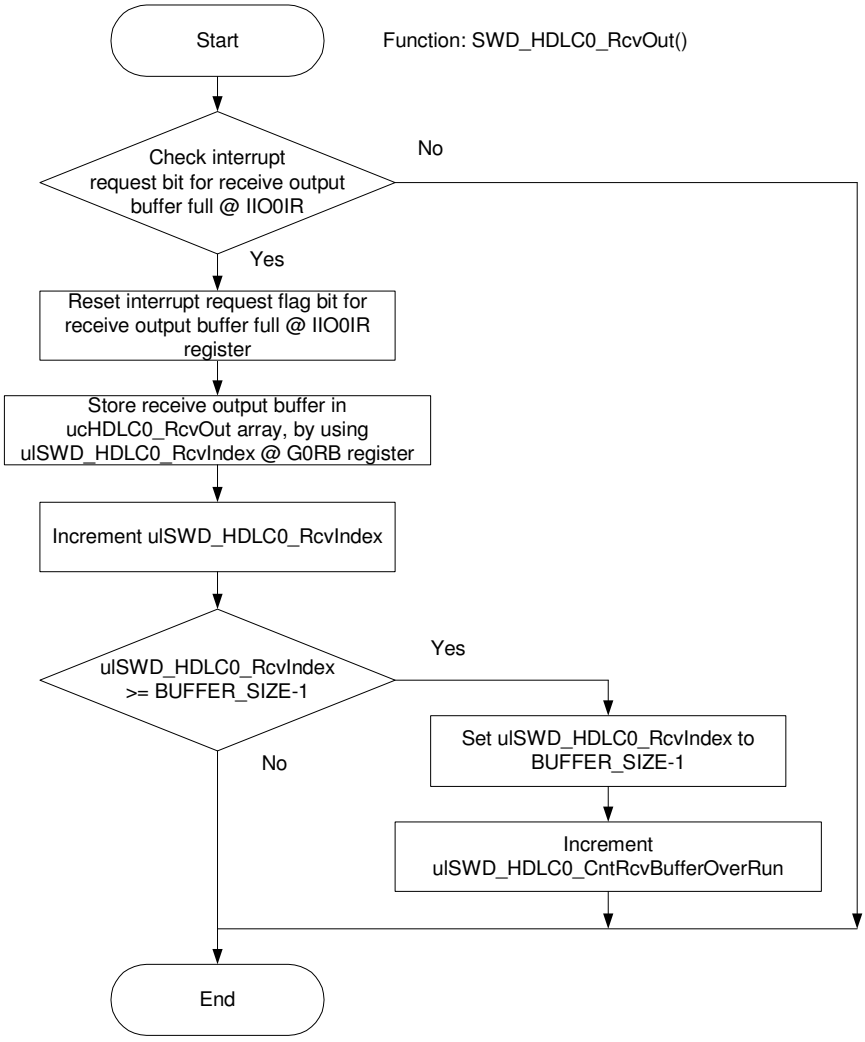
4.4.4 SWD_HDLC0_Init()



4.4.5 SWD_HDLC0_RcvIn() function

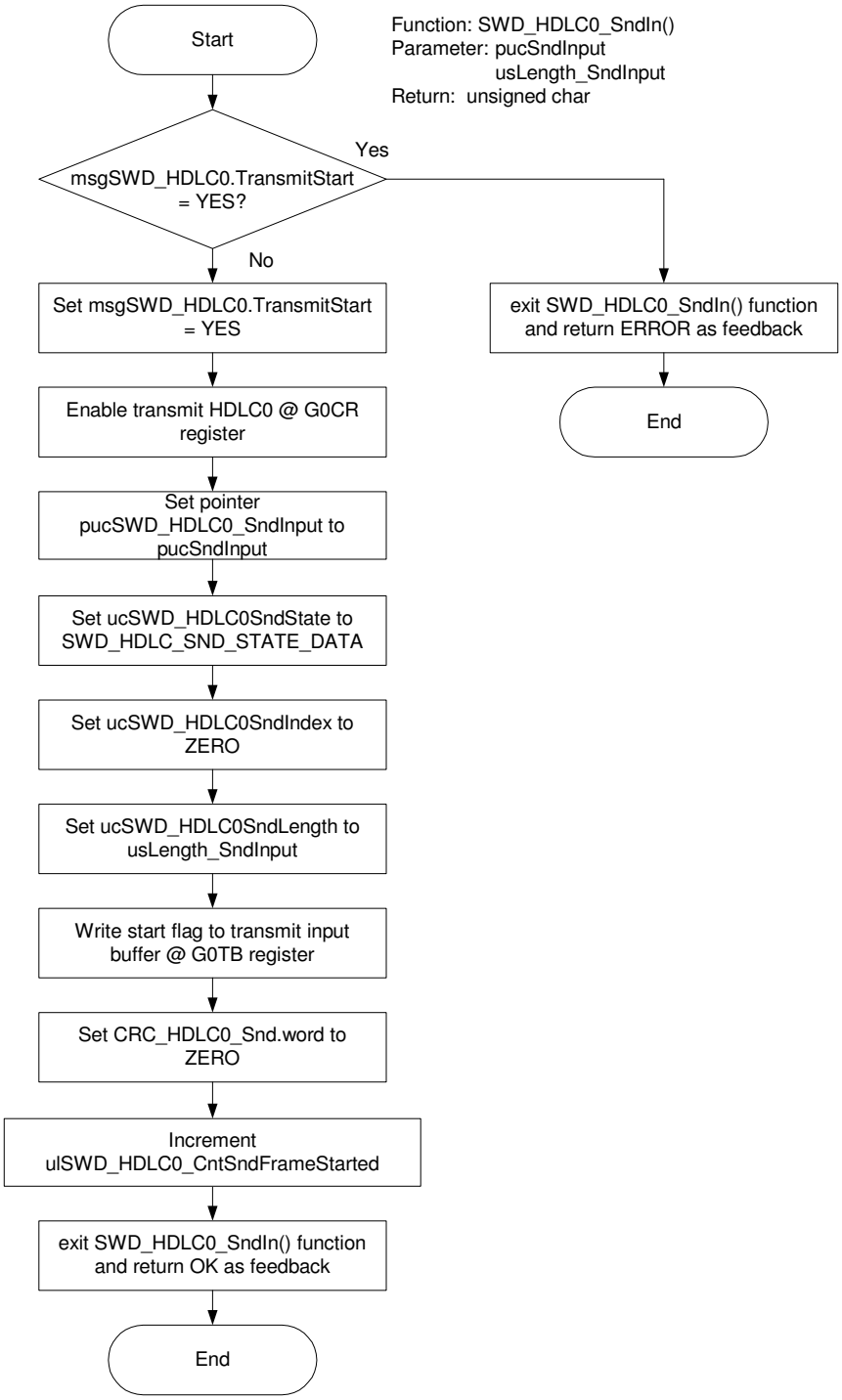


4.4.6 SWD_HDLC0_RcvOut() function

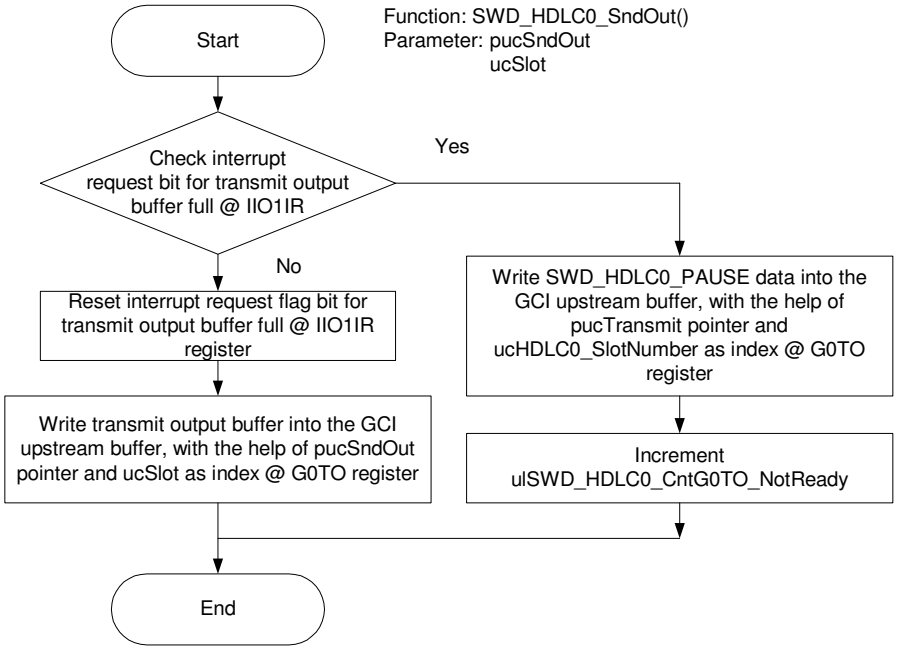


4.4.7 SWD_HDLC0_RcvPoll() function

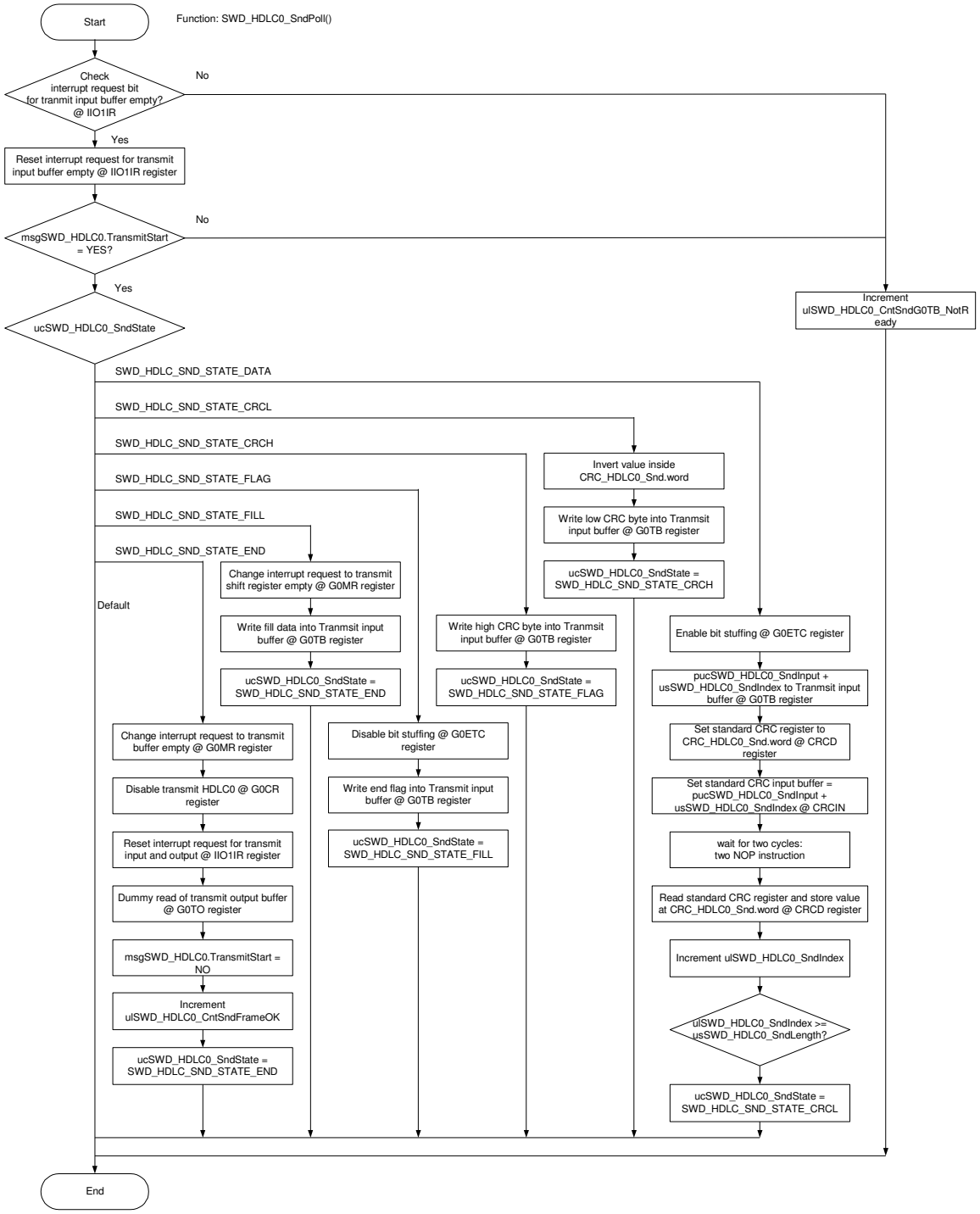
4.4.8 SWD_HDLC0_SndIn() function



4.4.9 SWD_HDLC0_SndOut() function



4.4.10 SWD_HDLC0_SndPoll() function



5 GCI-HDLC Driver C source code

This C source code supports IAR and Renesas compiler. To get a complete working project, a SFR header file and the related Cstartup files have to be added to these files to get a running project.

5.1 Main.c

```
/***********************************************************************/
/***********************************************************************/
/***************** Renesas Technology Europe GmbH **********************/
/***********************************************************************/
/***********************************************************************/
/*  DISCLAIMER:                                                        */
/*  We (Renesas Technology Europe GmbH) do not warrant that the Software  */
/*  is free from claims by a third party of copyright, patent, trademark, */
/*  trade secret or any other intellectual property infringement.      */
/*                                                                     */
/*  Under no circumstances are we liable for any of the following:     */
/*                                                                     */
/*  1. third-party claims against you for losses or damages;           */
/*  2. loss of, or damage to, your records or data; or                 */
/*  3. economic consequential damages (including lost profits or       */
/*     savings) or incidental damages, even if we are informed of      */
/*     their possibility.                                              */
/*                                                                     */
/*  We do not warrant uninterrupted or error free operation of the     */
/*  Software. We have no obligation to provide service, defect         */
/*  correction, or any maintenance for the Software. We have no        */
/*  obligation to supply any Software updates or enhancements to you   */
/*  even if such are or later become available.                        */
/*                                                                     */
/*  IF YOU DOWNLOAD OR USE THIS SOFTWARE YOU AGREE TO THESE TERMS.     */
/*                                                                     */
/*  THERE ARE NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING THE         */
/*  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A            */
/*  PARTICULAR PURPOSE.                                                */
/***********************************************************************/
/***********************************************************************/
/*                                                                     */
/*  Name:        main.c                                                */
/*  Part of:     M32C_GCI_HDLC_SoftwareDriver                          */
/*  Description: Main-modul                                            */
/*  Date  :      23.04.2003                                            */
/*  Author:      BWE @ Renesas Technology Europe GmbH                  */
/*  Change:      (Date)  (Author)  (Description)                       */
/*                                                                     */
/***********************************************************************/

#define EXTERN
#include    "sfr_3083.h"
#undef EXTERN

#define EXTERN extern
#include   "int_3083.h"
#include   "SWD_HDLC0.h"
#include   "SWD_HDLC1.h"
#include   "SWD_GCI.h"
#undef EXTERN

/***********************************************************************/
/************************ LOCAL HEADER START ***************************/
/***********************************************************************/
/************************** General Definitions ************************/
#define HDLC0               // Enable HDLC0 block
#define HDLC1               // Enable HDLC1 block

#define LENGTH_PARAMETER 0  // Indicates first byte of array, which actually
                            // contains the array length
```

```
#define FIRST_DATA_INDEX 1  // Indicates first data byte of array, which have
                            // to be transferred

/************************* Variable definition ***************************/
// Please Note:
// The content of following two arrays is just for testing purpose
//
/******************************** HDLC0 **********************************/
 unsigned char ucHDLC0_SndIn[]= {255,
  0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0A,0x0B,0x0C,0x0D,0x0E,0x0F,0x10,
  0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,0x19,0x1A,0x1B,0x1C,0x1D,0x1E,0x1F,0x20,
  0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,0x29,0x2A,0x2B,0x2C,0x2D,0x2E,0x2F,0x30,
  0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x3A,0x3B,0x3C,0x3D,0x3E,0x3F,0x40,
  0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x4A,0x4B,0x4C,0x4D,0x4E,0x4F,0x50,
  0x51,0x52,0x53,0x54,0x55,0x56,0x57,0x58,0x59,0x5A,0x5B,0x5C,0x5D,0x5E,0x5F,0x60,
  0x61,0x62,0x63,0x64,0x65,0x66,0x67,0x68,0x69,0x6A,0x6B,0x6C,0x6D,0x6E,0x6F,0x70,
  0x71,0x72,0x73,0x74,0x75,0x76,0x77,0x78,0x79,0x7A,0x7B,0x7C,0x7D,0x7E,0x7F,0x80,
  0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,
  0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x9C,0x9D,0x9E,0x9F,0xA0,
  0xA1,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xAF,0xB0,
  0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF,0xC0,
  0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,
  0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF,0xE0,
  0xE1,0xE2,0xE3,0xE4,0xE5,0xE6,0xE7,0xE8,0xE9,0xEA,0xEB,0xEC,0xED,0xEE,0xEF,0xF0,
  0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0xFF};

/******************************** HDLC1 **********************************/
 unsigned char ucHDLC1_SndIn[]= {255,
  0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0A,0x0B,0x0C,0x0D,0x0E,0x0F,0x10,
  0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,0x19,0x1A,0x1B,0x1C,0x1D,0x1E,0x1F,0x20,
  0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,0x29,0x2A,0x2B,0x2C,0x2D,0x2E,0x2F,0x30,
  0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x3A,0x3B,0x3C,0x3D,0x3E,0x3F,0x40,
  0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x4A,0x4B,0x4C,0x4D,0x4E,0x4F,0x50,
  0x51,0x52,0x53,0x54,0x55,0x56,0x57,0x58,0x59,0x5A,0x5B,0x5C,0x5D,0x5E,0x5F,0x60,
  0x61,0x62,0x63,0x64,0x65,0x66,0x67,0x68,0x69,0x6A,0x6B,0x6C,0x6D,0x6E,0x6F,0x70,
  0x71,0x72,0x73,0x74,0x75,0x76,0x77,0x78,0x79,0x7A,0x7B,0x7C,0x7D,0x7E,0x7F,0x80,
  0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,
  0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x9C,0x9D,0x9E,0x9F,0xA0,
  0xA1,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xAF,0xB0,
  0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF,0xC0,
  0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,
  0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF,0xE0,
  0xE1,0xE2,0xE3,0xE4,0xE5,0xE6,0xE7,0xE8,0xE9,0xEA,0xEB,0xEC,0xED,0xEE,0xEF,0xF0,
  0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0xFF};

/*************************  Function prototypes ***************************/
void Service_8KHZ(unsigned char * ,unsigned char * );

/*************************************************************************/
/************************ LOCAL HEADER END *******************************/
/*************************************************************************/

/*************************************************************************/
/*                                                                       */
/* Subroutine:  main                                                     */
/* Purpose:     main loop                                                */
/* Parameter:   No                                                       */
/* Return:      No                                                       */
/*                                                                       */
/*************************************************************************/
void main (void)
{
// PM0, PM0, CM0, CM1, MCD register is set in ncrt0.a30 or in cstartup.s48
//-----------------------------------------------------------------------
#ifdef HDLC0
  SWD_HDLC0_Init();           // Initialization of HDLC0
#endif // HDLC0
#ifdef HDLC1
  SWD_HDLC1_Init();           // Initialization of HDLC1
#endif // HDLC1

  SWD_GCI_Init(Service_8KHZ); // Call of init GCI function, parameter should be 8kHz routine
                              // 8kHz routine name can be adjust to user needs
  __enable_interrupt();       // enable all interrupts
```

```
  SWD_GCI_Start();            // Call of Start routine

  while (1)                   // endless while loop
  {
   asm("NOP");
  }
}

/************************************************************************/
/*                                                                      */
/* Subroutine:  Service_8KHZ                                            */
/* Purpose:     This routines is entered every 8kHz, due to call within */
/*              the DMA0 interrupt service routine. So this function call */
/*              is actually synchronous to 8kHz frequency of GCI FS     */
/*              signal.                                                 */
/*              The function name and syle can be adjusted to user needs, */
/*              but a new function name have to be refered to the GCI   */
/*              driver, by the SWD_GCI_Init(...) call.                  */
/*                                                                      */
/* Parameter:   pointer unsigned char transmit data array               */
/*              pointer unsigned char receive data array                */
/* Return:      No                                                      */
/*                                                                      */
/************************************************************************/
void Service_8KHZ ( unsigned char * pucTransmit, unsigned char * pucReceive)
{
unsigned char ucHDLC0_SlotNumber;  // number of slot for HDLC0
unsigned char ucHDLC1_SlotNumber;  // number of slot for HDLC1

#ifdef HDLC0
    ucHDLC0_SlotNumber = 0;  // a fix PCM slot is selected (0-31)
//-----------------------------------------------------------------------
// Recommended order for Receive HDLC0 handling:
// #1: Read G0RB
// #2: Check compare register (polling)
// #3: Write new data in G0RI
//-----------------------------------------------------------------------
  SWD_HDLC0_RcvOut();

  SWD_HDLC0_RcvPoll();  // Poll compare register

  SWD_HDLC0_RcvIn(pucReceive, ucHDLC0_SlotNumber);

//-----------------------------------------------------------------------
// Recommended order for Transmit HDLC0 handling:
// #1: Write G0TB (polling)
// #2: Write new data frame, if new frame is requested
// #3: Read data of G0TO
//-----------------------------------------------------------------------
  SWD_HDLC0_SndPoll();   // Poll transmit state machine

  // Write data frame which have to be transmitted by HDLC, by calling following function
  // including pointer to data array plus array length
  SWD_HDLC0_SndIn(&ucHDLC0_SndIn[FIRST_DATA_INDEX], ucHDLC0_SndIn[LENGTH_PARAMETER]);

  SWD_HDLC0_SndOut(pucTransmit, ucHDLC0_SlotNumber);

//-----------------------------------------------------------------------
#endif //HDLC0

#ifdef HDLC1
    ucHDLC1_SlotNumber = 5;  // a fix PCM slot is selected (0-31)
//-----------------------------------------------------------------------
// Recommended order for Receive HDLC1 handling:
// #1: Read G1RB
// #2: Check compare register (polling)
// #3: Write new data in G1RI
//-----------------------------------------------------------------------
  SWD_HDLC1_RcvOut();

  SWD_HDLC1_RcvPoll();  // Poll compare register

  SWD_HDLC1_RcvIn(pucReceive, ucHDLC1_SlotNumber);

//-----------------------------------------------------------------------
```

```
// Recommended order for Transmit HDLC1 handling:
// #1: Write G1TB (polling)
// #2: Write new data frame, if new frame is requested
// #3: Read data of G1TO
//-----------------------------------------------------------------------
  SWD_HDLC1_SndPoll();  // Poll transmit state machine

  // Write data frame which have to be transmitted by HDLC, by calling following function
  // including pointer to data array plus array length
  SWD_HDLC1_SndIn(&ucHDLC1_SndIn[FIRST_DATA_INDEX], ucHDLC1_SndIn[LENGTH_PARAMETER]);

  SWD_HDLC1_SndOut(pucTransmit, ucHDLC1_SlotNumber);
//-----------------------------------------------------------------------
#endif //HDLC1

}
/********************************** E O F ******************************/
```

5.2 SWD_GCI.h

```
/*****************************************************************************/
/*****************************************************************************/
/****************** Renesas Technology Europe GmbH ***********************/
/*****************************************************************************/
/*****************************************************************************/
/*  DISCLAIMER:                                                              */
/*  We (Renesas Technology Europe GmbH) do not warrant that the Software     */
/*  is free from claims by a third party of copyright, patent, trademark,    */
/*  trade secret or any other intellectual property infringement.            */
/*                                                                           */
/*  Under no circumstances are we liable for any of the following:           */
/*                                                                           */
/*  1. third-party claims against you for losses or damages;                 */
/*  2. loss of, or damage to, your records or data; or                       */
/*  3. economic consequential damages (including lost profits or             */
/*     savings) or incidental damages, even if we are informed of            */
/*     their possibility.                                                    */
/*                                                                           */
/*  We do not warrant uninterrupted or error free operation of the           */
/*  Software. We have no obligation to provide service, defect               */
/*  correction, or any maintenance for the Software. We have no              */
/*  obligation to supply any Software updates or enhancements to you         */
/*  even if such are or later become available.                              */
/*                                                                           */
/*  IF YOU DOWNLOAD OR USE THIS SOFTWARE YOU AGREE TO THESE TERMS.           */
/*                                                                           */
/*  THERE ARE NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING THE               */
/*  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A                  */
/*  PARTICULAR PURPOSE.                                                      */
/*****************************************************************************/
/*****************************************************************************/
/*                                                                           */
/*  Name:        SWD_GCI.h                                                   */
/*  Part of:     M32C_GCI_SoftwareDriver                                     */
/*  Description: global definitions + declarations for GCI driver module     */
/*  Date  :      23.04.2003                                                  */
/*  Author:      BWE @ Renesas Technology Europe GmbH                        */
/*  Change:      (Date)  (Author)  (Description)                             */
/*                                                                           */
/*****************************************************************************/

/*****************************************************************************/
/************************** GLOBAL HEADER PART *******************************/
/*****************************************************************************/
/************************  Function prototypes *******************************/
EXTERN void SWD_GCI_Init (void (*)());
EXTERN void SWD_GCI_Start (void);
EXTERN void SWD_GCI_Stop (void);

/********************************** E O F ******************************/
```

5.3 SWD_GCI.c

```
/*****************************************************************************/
/*****************************************************************************/
/****************** Renesas Technology Europe GmbH ***************************/
/*****************************************************************************/
/*****************************************************************************/
/*  DISCLAIMER:                                                              */
/*  We (Renesas Technology Europe GmbH) do not warrant that the Software     */
/*  is free from claims by a third party of copyright, patent, trademark,    */
/*  trade secret or any other intellectual property infringement.            */
/*                                                                           */
/*  Under no circumstances are we liable for any of the following:           */
/*                                                                           */
/*  1. third-party claims against you for losses or damages;                 */
/*  2. loss of, or damage to, your records or data; or                       */
/*  3. economic consequential damages (including lost profits or             */
/*     savings) or incidental damages, even if we are informed of            */
/*     their possibility.                                                    */
/*                                                                           */
/*  We do not warrant uninterrupted or error free operation of the           */
/*  Software. We have no obligation to provide service, defect               */
/*  correction, or any maintenance for the Software. We have no              */
/*  obligation to supply any Software updates or enhancements to you         */
/*  even if such are or later become available.                              */
/*                                                                           */
/*  IF YOU DOWNLOAD OR USE THIS SOFTWARE YOU AGREE TO THESE TERMS.           */
/*                                                                           */
/*  THERE ARE NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING THE               */
/*  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A                  */
/*  PARTICULAR PURPOSE.                                                      */
/*****************************************************************************/
/*****************************************************************************/
/*                                                                           */
/*  Name:        SWD_GCI.c                                                   */
/*  Part of:     M32C_GCI_SoftwareDriver                                     */
/*  Description: GCI driver module                                           */
/*  Date  :      23.04.2003                                                  */
/*  Author:      BWE @ Renesas Technology Europe GmbH                        */
/*  Change:      (Date)  (Author)  (Description)                             */
/*                                                                           */
/*****************************************************************************/
#define EXTERN extern
#include   "sfr_3083.h"
#include   "int_3083.h"
#undef EXTERN

#define EXTERN
#include    "SWD_GCI.h"
#undef  EXTERN

/*****************************************************************************/
/************************** LOCAL HEADER START *******************************/
/*****************************************************************************/
/************************** Definitions **************************************/
/* Only one mode have to be selected */
//#define GCI_MODE_TE
//#define GCI_MODE_NT
#define GCI_MODE_PCM

#ifdef GCI_MODE_TE
#define SWD_GCI_SLOT 12  // Number of channels per frame for PCM mode
#endif

#ifdef GCI_MODE_NT
#define SWD_GCI_SLOT 32  // Number of channels per frame for PCM mode
#endif

#ifdef GCI_MODE_PCM
#define SWD_GCI_SLOT 32  // Number of channels per frame for PCM mode
#endif

/* Only one synch behavior has to be selected */
#define SYNCH_AT_RISING_DCL
```

```
//#define SYNCH_AT_FALLING_DCL

/************************ Function prototypes ****************************/
void SWD_GCI_SetRcv (void);
void SWD_GCI_SetSnd (void);
void (* SWD_GCI_CallBackFunction) ();

/************************ variable definition ****************************/
unsigned char ucSWD_GCI_RcvBuffer[SWD_GCI_SLOT*2];  // definition of 2 buffers for receive
unsigned char ucSWD_GCI_SndBuffer[SWD_GCI_SLOT*2];  // definition of 2 buffers for transmit

unsigned char * pucSWD_GCI_UartToBuf;  // pointer to receive buffer
unsigned char * pucSWD_GCI_BufToMcu;   // pointer to receive buffer

unsigned char * pucSWD_GCI_BufToUart;  // pointer to transmit buffer
unsigned char * pucSWD_GCI_McuToBuf;   // pointer to transmit buffer

unsigned char * pucSWD_GCI_ToggleBuffer; // help pointer for buffer toggle

/*************************************************************************/
/************************ LOCAL HEADER END *******************************/
/*************************************************************************/

/*************************************************************************/
/*                                                                       */
/* Subroutine:  SWD_GCI_Init                                             */
/* Purpose:     Initialization of UART2 in GCI mode and related port     */
/*              setting                                                  */
/* Parameter:   pointer to function within the user software, which is   */
/*              entered, due to a call within the DMA0 interrupt routine */
/* Return:      No                                                       */
/*                                                                       */
/*************************************************************************/
void SWD_GCI_Init (void (*ServiceFunction)())
{
   /* Setting of port register */
   P7 |= 0x01;           // Set GCI TxD output at UART2 to high, to ensure
                         // high level at GCI transmit line
   PD7 |= 0x01;          // Set GCI TxD at UART2 direction to output

   /* Setting of port function select register */
   PS1 |= 0x01;          // set port 70 peripheral function enable bit
   PSL1 &= ~0x01;        // reset port 70 TxD2 output
   PSC  &= ~0x01;        // Port70 output peripheral function select bit
   PS1 &= ~0x04;         // Port 70 peripheral function enable bit

   SWD_GCI_CallBackFunction = ServiceFunction;  // Transfer address parameter
                                                // for a later function call at
                                                // this address

   /* Call of init GCI transmit function */
   SWD_GCI_SetSnd();

   /* Call of init GCI receive function */
   SWD_GCI_SetRcv();

   /* Setting UART2 transmit/receive mode register */
   U2MR = 0x09;    // X--- XXXX
                   // |    |||+- Synchonous serial mode
                   // |    ||+-- Synchonous serial mode
                   // |    |+--- Synchonous serial mode
                   // |    |     Must be fixed to 001
                   // |    +---- Internal/external clock select bit
                   // |          0: Internal clock
                   // |          1: External clock
                   // +--------- TxD, RxD I/O polartiy reverse bit

   /* Setting UART2 transmit/receive control register 0 */
   #ifdef GCI_MODE_PCM
    #ifdef SYNCH_AT_RISING_DCL
    U2C0 = 0xF0;   // XX-X XXXX  PCM mode use MSB first
    #endif
    #ifdef SYNCH_AT_FALLING_DCL
    U2C0 = 0xB0;   // XX-X XXXX  PCM mode use MSB first
    #endif
```

```
#else
U2C0 = 0x70;    // XX-X XXXX  TE and NT mode uses LSB first
#endif
                // || | |||+- BRG count source select bit
                // || | ||+-- BRG count source select bit
                // || | ||    00: f1 is selected
                // || | ||    01: f8 is selected
                // || | ||    10: f32 is selected
                // || | ||    11: inhibited
                // || | |+--- /CTS//RTS function select bit
                // || | |     (Valid when bit 4 ='0')
                // || | |     0: /CTS function is selected
                // || | |     1: /RTS function is selected
                // || | +---- Transmit register empty flag
                // || |       0: Data present in transmit register
                // || |          (during transmission)
                // || |       1: No Data present in transmit register
                // || |          (transmission completed)
                // || +------ /CTS//RTS disable bit
                // ||         0: /CTS//RTS function enabled
                // ||         1: /CTS//RTS function disenabled
                // |+-------- CLK polatity select bit
                // |          0: Transmit data is output at falling
                // |             edge of transfer clock and receive
                // |             data is input at rising edge
                // |          1: Transmit data is output at rising
                // |             edge of transfer clock and receive
                // |             data is input at falling edge
                // +--------- Transfer format select bit
                //            0: LSB first
                //            1: MSB first

/* Setting UART special mode register */
#ifdef GCI_MODE_PCM
U2SMR &= ~0x80; // XXXX XXXX  PCM mode use MSB first
#else
U2SMR |= 0x80;  // XXXX XXXX  TE and NT mode uses divided clock
#endif
                // |||| |||+- IICM mode select bit
                // |||| |||   0: Normal mode
                // |||| |||   1: IIC mode
                // |||| ||+-- Arbitration lost detecting flag control bit
                // |||| ||    0: Update per bit
                // |||| ||    1: Update per byte
                // |||| |+--- Bus busy flag
                // |||| |     0: Stop condition deteced
                // |||| |     1: Start condition detected
                // |||| +---- SCLL sync output enable bit
                // ||||       0: Disabled
                // ||||       1: Enabled
                // |||+------ Bus collision detect sampling clock select bit
                // |||        Set to "0"
                // ||+------- Auto clear function of transmit enable bit
                // ||         Set to "0"
                // |+-------- Transmit start condition select bit
                // |          Set to "0"
                // +--------- Clock divide set bit
                //            0: Not divided
                //            1: Divided

/* Setting UART special mode register 2 */
#ifdef GCI_MODE_PCM
U2SMR2 |= 0x80; // XXXX XXXX  PCM clock is synchronous to external clock
#else
U2SMR2 &= ~0x80;// XXXX XXXX  TE and NT clock is not synchronous to external clock
#endif
                // |||| |||+- IIC mode select bit 2
                // |||| |||   0: NACK/ACK interrupt (DMA source -ACK)
                // |||| |||   1: UART transfer/receive interrupt
                // |||| |||      (DMA source A1: -UART receive)
                // |||| ||+-- Clock synchronous bit
                // |||| ||    0: Disabled
                // |||| ||    1: Enabled
                // |||| |+--- SCL wait output bit
                // |||| |     0: Disabled
```

```
                    // |||| |     1: Enabled
                    // |||| +---- SDA output stop bit
                    // ||||       0: Disabled
                    // ||||       1: Enabled
                    // |||+------ UARTi initialize bit
                    // |||        0: Disabled
                    // |||        1: Enabled
                    // ||+------- SCL wait output bit 2
                    // ||         0: UARTi clock
                    // ||         1: 0 output
                    // |+-------- SDA output inhibit bit
                    // |          0: Disabled
                    // |          1: Enabled
                    // +--------- External clock synchronizing enable bit
                    //            0: Synchronous disabled
                    //            1: Synchronous enabled
}

/************************************************************************/
/*                                                                      */
/* Subroutine:  SWD_GCI_Start                                           */
/* Purpose:     Start GCI reception and transmission process            */
/* Parameter:   No                                                      */
/* Return:      No                                                      */
/*                                                                      */
/************************************************************************/
void SWD_GCI_Start ()
{
   /* Re-synchronisation before start up GCI communication*/
   U2C1 &= ~0x80;       // Re-synchronization with FS signal
   U2C1 &= ~0x05;       // prohibit receive and transmit
   U2MR &= ~0x07;       // invalidate serial I/O mode
   U2MR |= 0x01;        // set up serial I/O mode

 //  U2C1 |= 0x80;

   U2TB = 0xFF;     // write dummy data into transmit buffer

   asm("LDC #0B3H, DMD0");  // enable DMAs in repeat mode

   U2C1 |= 0x05;            // enable receive and transmit
}

/************************************************************************/
/*                                                                      */
/* Subroutine:  SWD_GCI_Stop                                            */
/* Purpose:     Stop GCI reception and transmission process             */
/* Parameter:   No                                                      */
/* Return:      No                                                      */
/*                                                                      */
/************************************************************************/
void SWD_GCI_Stop ()
{
   asm("LDC #00H, DMD0");   // disable DMAs in any mode

   U2C1 &= ~0x05;           // disable receive and transmit

   SWD_GCI_Init(SWD_GCI_CallBackFunction); // Call Init GCI routine to be
                                           // prepared for a new SWD_GCI_Start
                                           // call
}

/************************************************************************/
/*                                                                      */
/* Subroutine:  SWD_GCI_SetRcv                                          */
/* Purpose:     Initialize DMA0 for receive GCI data stream handling    */
/* Parameter:   No                                                      */
/* Return:      No                                                      */
/*                                                                      */
/************************************************************************/
void SWD_GCI_SetRcv ()
{
   unsigned short volatile * pusSource; // source address
   unsigned char * pucDestination;      // destination address
   unsigned char * pucDestination2;     // destination address
```

```
  unsigned char ucLoop;   // loop variable

  /* Set DMA mode register DMD0, disable DMA0 and DMA1 */
  asm("LDC #00H,DMD0");   // XXXX XXXX
                          //      ||++- Channel 0 transfer mode select bit
                          //      ||    00: DMA inhibit
                          //      ||    01: Single transfer
                          //      ||    10: Reserved
                          //      ||    11: Reapeat transfer
                          //      |+--- Cannel 0 transfer direction select bit
                          //      |     0: 8 bits
                          //      |     1: 16 bits
                          //      +---- Channel 0 transfer direction select bit
                          //            0: Fixed address to memory
                          //            1: Memory to fixed address

  /* set DMA0 request cause register */
  DM0SL = 0x13;           // X-XX XXXX
                          // | || ||||  DMA
                          // | |+-++++- DMA request cause select bit
                          // | |        10011: UART2 receive/ACK
                          // | +------- Software DMA request bit
                          // |          If Software trigger is selected a
                          // |          DMA request is generated by setting
                          // |          this bit to "1"
                          // +--------- DMA request bit
                          //            0: Not requested
                          //            1: Requested

  /* initialize receive buffer */
  for(ucLoop=0; ucLoop < (SWD_GCI_SLOT*2) ; ucLoop++)
  {
     ucSWD_GCI_RcvBuffer[ucLoop] = 0xFF; // set all buffer to ones
  }

  pucSWD_GCI_UartToBuf = &(ucSWD_GCI_RcvBuffer[0]); // set one pointer to start of buffer
  pucSWD_GCI_BufToMcu = &(ucSWD_GCI_RcvBuffer[SWD_GCI_SLOT]);  // set one pointer to mid of buffer

  pucDestination = pucSWD_GCI_UartToBuf;    // Set destination to transmit Buffer 0
  pucDestination2 = pucSWD_GCI_BufToMcu;    // Set destination to transmit Buffer 1

  pusSource = &U2RB;                        // Set source to UART2 receive buffer

 #ifdef NC308 // If Renesas Compiler is selected
  asm("LDC $$[FB], DMA0", pucDestination);  // Set DMA memory address register
                                            // Set destination to Receive Buffer 0
  asm("LDC $$[FB], DSA0", pusSource);       // Set origin to Uart2
  asm("LDC $$[FB], DRA0", pucDestination2); // Set DMA memory reload address register
                                            // Set destination to Receive Buffer 1
 #endif //NC308

 #ifdef __IAR_SYSTEMS_ICC__   // If IAR Compiler is selected
  __intrinsic_load_DMA(0,(unsigned long)pucDestination);  // Set DMA memory address register
                                                          // Set destination to Receive Buffer 0
  __intrinsic_load_DSA(0,(unsigned long)pusSource);       // Set origin to Uart2
  __intrinsic_load_DRA(0,(unsigned long)pucDestination2); // Set DMA memory reload address register
 #endif //__IAR_SYSTEMS_ICC__

  #ifdef GCI_MODE_TE
   asm("LDC #0CH, DCT0");     // Set DMA transfer count register 12 byte
   asm("LDC #0CH, DRC0");     // Set DMA transfer count reload register 12 byte
  #else
   asm("LDC #20H, DCT0");     // Set DMA transfer count register 32 byte
   asm("LDC #20H, DRC0");     // Set DMA transfer count reload register 32 byte
  #endif

  /* Set DMA mode register DMD0, enable dma0 interrupt */
  DM0IC = 0x07;           // ---- XXXX
                          //      ||||
                          //      |||+-- Interupt priority level select bit
                          //      ||+--- Interupt priority level select bit
                          //      |+---- Interupt priority level select bit
                          //      |       000: Level 0 (interrupt disabled)
                          //      |       001: Level 1
                          //      |       010: Level 2
```

```
                        //      |      011: Level 3
                        //      |      100: Level 4
                        //      |      101: Level 5
                        //      |      110: Level 6
                        //      |      111: Level 7
                        //      +----- Interupt request bit
                        //             0: Interrupt not requested
                        //             1: Interrupt requested

   /*-------------Swap buffer pointers----------*/
   /*---------------receive buffers------------*/
   pucSWD_GCI_ToggleBuffer = pucSWD_GCI_UartToBuf;
   pucSWD_GCI_UartToBuf = pucSWD_GCI_BufToMcu;
   pucSWD_GCI_BufToMcu = pucSWD_GCI_ToggleBuffer;
}

/*****************************************************************************/
/*                                                                           */
/* Subroutine:  SWD_GCI_SetSnd                                               */
/* Purpose:     Initialize DMA1 for transmit GCI data stream handling        */
/* Parameter:   No                                                           */
/* Return:      No                                                           */
/*                                                                           */
/*****************************************************************************/
void SWD_GCI_SetSnd ()
{
   unsigned char * pucSource;        // source address
   unsigned char * pucSource2;       // source address
   unsigned short volatile * pusDestination;   // destination address
   unsigned char ucLoop;   // loop variable

   /* Set DMA mode register DMD0, disable DMA0 */
   asm("LDC #00H, DMD0");  // XXXX XXXX
                           //      ||++- Channel 0 transfer mode select bit
                           //      ||    00: DMA inhibit
                           //      ||    01: Single transfer
                           //      ||    10: Reserved
                           //      ||    11: Reapeat transfer
                           //      |+--- Cannel 0 transfer direction select bit
                           //      |     0: 8 bits
                           //      |     1: 16 bits
                           //      +---- Channel 0 transfer direction select bit
                           //            0: Fixed address to memory
                           //            1: Memory to fixed address

   /* set DMA1 request cause register */
   DM1SL    =  0x12;       // X-XX XXXX
                           // | || ||||  DMA
                           // | |+-++++- DMA request cause select bit
                           // | |        10010: UART2 transmit
                           // | |
                           // | +------- Software DMA request bit
                           // |          If Software trigger is selected a
                           // |          DMA request is generated by setting
                           // |          this bit to "1"
                           // +--------- DMA request bit
                           //            0: Not requested
                           //            1: Requested

   /* initialize transmit buffer */
   for(ucLoop=0; ucLoop < (SWD_GCI_SLOT*2) ; ucLoop++)
   {
     ucSWD_GCI_SndBuffer[ucLoop] = 0xFF; // set buffer to one
   }

   pucSWD_GCI_BufToUart = &(ucSWD_GCI_SndBuffer[0]); // set one pointer to start of buffer
   pucSWD_GCI_McuToBuf = &(ucSWD_GCI_SndBuffer[SWD_GCI_SLOT]);  // set one pointer to mid of buffer

   pucSource = (pucSWD_GCI_BufToUart+1);      // Set origin to transmit Buffer 0
   pucSource2 = pucSWD_GCI_McuToBuf;          // Set origin to transmit Buffer 1

   pusDestination = &(U2TB);                  // Set destination to UART2 transmit buffer

  #ifdef NC308 // If Renesas Compiler is selected
   asm("LDC $$[FB], DMA1", pucSource);       // Set DMA memory address register (Buffer 0)
```

```
   asm("LDC $$[FB], DSA1", pusDestination);  // Set destination to Uart2
   asm("LDC $$[FB], DRA1", pucSource2);      // Set DMA memory reload address register
  #endif // NC308

  #ifdef __IAR_SYSTEMS_ICC__   // If IAR Compiler is selected
   __intrinsic_load_DMA(1,(unsigned long)pucSource);      // Set DMA memory address register (Buffer 0)
   __intrinsic_load_DSA(1,(unsigned long)pusDestination);  // Set destination to Uart2
   __intrinsic_load_DRA(1,(unsigned long)pucSource2);      // Set DMA memory reload address register
  #endif // __IAR_SYSTEMS_ICC__

   #ifdef GCI_MODE_TE
    asm("LDC #0BH, DCT1");   // Set DMA transfer count register 11 byte
    asm("LDC #0CH, DRC1");   // Set DMA transfer count reload register 12 byte
   #else
    asm("LDC #1FH, DCT1");   // Set DMA transfer count register 31 byte
    asm("LDC #20H, DRC1");   // Set DMA transfer count reload register 32 byte
   #endif

   /*-------------Swap buffer pointers----------*/
   /*---------------transmit buffers------------*/
   pucSWD_GCI_ToggleBuffer = pucSWD_GCI_BufToUart;
   pucSWD_GCI_BufToUart = pucSWD_GCI_McuToBuf;
   pucSWD_GCI_McuToBuf = pucSWD_GCI_ToggleBuffer;
}

/*****************************************************************************/
/*                                                                           */
/* Interrupt:   SWD_GCI_RcvSndRdy                                            */
/* Purpose:     DMA0 interrupt routine, GCI frame received/sent              */
/*              because receive and transmit is the same length, one         */
/*              interrupt routine is sufficient.                             */
/*              This routine occur every 8kHz, due to GCI FS signal          */
/*              and 32 byte data periode.                                    */
/* Parameter:   No                                                           */
/* Return:      No                                                           */
/*                                                                           */
/*****************************************************************************/
__interrupt void SWD_GCI_RcvSndRdy (void)
{
   /* Call of 8KHZ Service routine with indirect pointer to user function */
   /* Parameter: pointer transmit buffer, pointer receive buffer */
   (* SWD_GCI_CallBackFunction) (pucSWD_GCI_McuToBuf, pucSWD_GCI_BufToMcu);

   /*-------------Swap buffer pointers----------*/
   /*---------------transmit buffers------------*/
   pucSWD_GCI_ToggleBuffer = pucSWD_GCI_BufToUart;
   pucSWD_GCI_BufToUart = pucSWD_GCI_McuToBuf;
   pucSWD_GCI_McuToBuf = pucSWD_GCI_ToggleBuffer;
   /* Set DMA1 memory reload address register */
  #ifdef NC308 // If Renesas Compiler is selected
   asm("LDC $$[FB], DRA1", pucSWD_GCI_BufToUart);
  #endif //NC308
  #ifdef __IAR_SYSTEMS_ICC__  // If IAR Compiler is selected
   __intrinsic_load_DRA(1,(unsigned long)pucSWD_GCI_BufToUart);
  #endif // __IAR_SYSTEMS_ICC__

   /*-------------Swap buffer pointers----------*/
   /*---------------receive buffers-------------*/
   pucSWD_GCI_ToggleBuffer = pucSWD_GCI_UartToBuf;
   pucSWD_GCI_UartToBuf = pucSWD_GCI_BufToMcu;
   pucSWD_GCI_BufToMcu = pucSWD_GCI_ToggleBuffer;
   /* Set DMA0 memory reload address register */
  #ifdef NC308 // If Renesas Compiler is selected
   asm("LDC $$[FB], DRA0", pucSWD_GCI_UartToBuf);
  #endif //NC308
  #ifdef __IAR_SYSTEMS_ICC__  // If IAR Compiler is selected
   __intrinsic_load_DRA(0,(unsigned long)pucSWD_GCI_UartToBuf);
  #endif // __IAR_SYSTEMS_ICC__
}

/********************************** E O F ************************************/
```

5.4 SWD_HDLC0.h

```
/*****************************************************************************/
/*****************************************************************************/
/****************** Renesas Technology Europe GmbH ***************************/
/*****************************************************************************/
/*****************************************************************************/
/*  DISCLAIMER:                                                              */
/*  We (Renesas Technology Europe GmbH) do not warrant that the Software     */
/*  is free from claims by a third party of copyright, patent, trademark,    */
/*  trade secret or any other intellectual property infringement.            */
/*                                                                           */
/*  Under no circumstances are we liable for any of the following:           */
/*                                                                           */
/*  1. third-party claims against you for losses or damages;                 */
/*  2. loss of, or damage to, your records or data; or                       */
/*  3. economic consequential damages (including lost profits or             */
/*     savings) or incidental damages, even if we are informed of            */
/*     their possibility.                                                    */
/*                                                                           */
/*  We do not warrant uninterrupted or error free operation of the           */
/*  Software. We have no obligation to provide service, defect               */
/*  correction, or any maintenance for the Software. We have no              */
/*  obligation to supply any Software updates or enhancements to you         */
/*  even if such are or later become available.                              */
/*                                                                           */
/*  IF YOU DOWNLOAD OR USE THIS SOFTWARE YOU AGREE TO THESE TERMS.           */
/*                                                                           */
/*  THERE ARE NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING THE               */
/*  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A                  */
/*  PARTICULAR PURPOSE.                                                      */
/*****************************************************************************/
/*****************************************************************************/
/*                                                                           */
/*  Name:        SWD_HDLC0.h                                                 */
/*  Part of:     M32C_HDLC_SoftwareDriver                                    */
/*  Description: global definitions and declarations for HDLC function       */
/*  Date  :      23.04.2003                                                  */
/*  Author:      BWE @ Renesas Technology Europe GmbH                        */
/*  Change:      (Date)  (Author)  (Description)                             */
/*                                                                           */
/*****************************************************************************/

/*****************************************************************************/
/*************************** GLOBAL HEADER PART ******************************/
/*****************************************************************************/
/***************************** General makros ********************************/
#define SWD_HDLC0_OK     0x00
#define SWD_HDLC0_ERROR  0x01

#define SWD_HDLC0_NO 0
#define SWD_HDLC0_YES 1

/************************** Function prototypes ******************************/
// HDLC0 initialization
EXTERN void SWD_HDLC0_Init(void);
// Function which have to be polled by user to start transmission
EXTERN unsigned char SWD_HDLC0_SndIn(unsigned char *, unsigned short);
// Function which have to be polled by user for transmission output
EXTERN void SWD_HDLC0_SndOut(unsigned char *, unsigned char);
// Function which have to be polled by user for transmission
EXTERN void SWD_HDLC0_SndPoll(void);
// Function which have to be polled by user for reception
EXTERN void SWD_HDLC0_RcvPoll(void);
// Function which have to be polled by user for reception input
EXTERN void SWD_HDLC0_RcvIn(unsigned char *, unsigned char);
// Function which have to be polled by user for reception output
EXTERN void SWD_HDLC0_RcvOut(void);
/************************** Variable Definition ******************************/

/*********************************** E O F ***********************************/
```

5.5 SWD_HDLC0.c

```
/*****************************************************************************/
/*****************************************************************************/
/****************** Renesas Technology Europe GmbH ***************************/
/*****************************************************************************/
/*****************************************************************************/
/*  DISCLAIMER:                                                              */
/*  We (Renesas Technology Europe GmbH) do not warrant that the Software     */
/*  is free from claims by a third party of copyright, patent, trademark,    */
/*  trade secret or any other intellectual property infringement.            */
/*                                                                           */
/*  Under no circumstances are we liable for any of the following:           */
/*                                                                           */
/*  1. third-party claims against you for losses or damages;                 */
/*  2. loss of, or damage to, your records or data; or                       */
/*  3. economic consequential damages (including lost profits or             */
/*     savings) or incidental damages, even if we are informed of            */
/*     their possibility.                                                    */
/*                                                                           */
/*  We do not warrant uninterrupted or error free operation of the           */
/*  Software. We have no obligation to provide service, defect               */
/*  correction, or any maintenance for the Software. We have no              */
/*  obligation to supply any Software updates or enhancements to you         */
/*  even if such are or later become available.                              */
/*                                                                           */
/*  IF YOU DOWNLOAD OR USE THIS SOFTWARE YOU AGREE TO THESE TERMS.           */
/*                                                                           */
/*  THERE ARE NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING THE               */
/*  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A                  */
/*  PARTICULAR PURPOSE.                                                      */
/*****************************************************************************/
/*****************************************************************************/
/*                                                                           */
/*  Name:        SWD_HDLC0.c                                                 */
/*  Part of:     M32C_HDLC_SoftwareDriver                                    */
/*  Description: HDLC driver module                                          */
/*  Date  :      23.04.2003                                                  */
/*  Author:      BWE @ Renesas Technology Europe GmbH                        */
/*  Change:      (Date)  (Author)  (Description)                             */
/*                                                                           */
/*****************************************************************************/
#define EXTERN extern
#include   "sfr_3083.h"
#include   "int_3083.h"
#undef EXTERN

#define EXTERN
#include "SWD_HDLC0.h"
#undef EXTERN

/*****************************************************************************/
/************************* LOCAL HEADER START ********************************/
/*****************************************************************************/
/* Only one transmit pause mode have to be selected */
//#define MARK_IDLE       // constantly 1 characters as pause signal
#define FLAG_IDLE         // flag characters as pause signal

/************************* Definitions ***********************************/
#define SWD_HDLC0_GOOD_CRC   0x0B9F  // Result for correct CRC reception
                                     // (includes flag)
#define SWD_HDLC0_FLAG       0x7E    // Start / end flag
#define SWD_HDLC0_ABORT      0xFE    // Abort 11111110
#define SWD_HDLC0_ABORT_MASK 0x01    // 1st bit mask for abort
#define SWD_HDLC0_FILL       0xFF    // Data between frames

#ifdef MARK_IDLE
 #define SWD_HDLC0_PAUSE      0xFF   // Ones as pause signal for HDLC traffic
#endif

#ifdef FLAG_IDLE
 #define SWD_HDLC0_PAUSE      0x7E   // flag characters as pause signal
#endif
```

```
#define SWD_HDLC0_FRAME_MIN  3       // Set minimum frame size

#define SWD_HDLC0_CRC_INIT 0xFFFF // Init value for CRC generation

#define ZERO 0

#define BUFFER_SIZE 1001    // Receive output buffer size definition

/************************ Function prototypes ****************************/
void SWD_HDLC0_Basetimer_Init(void);    // Initialisation of Basetimer0

/************************ Message structure ******************************/
typedef struct
  {
    unsigned short TransmitStart: 1; // 1: Process for transmission frame has been started
    unsigned short ReceiveFlagDetected: 1; // 1: Flag Detected
    /*  Place for more messages  */
  } TSWD_HDLC0_Msg;

TSWD_HDLC0_Msg msgSWD_HDLC0;

/****************** States of transmit state machine **********************/
enum SWD_HDLC_SndEnum
      {
          SWD_HDLC_SND_STATE_DATA,
          SWD_HDLC_SND_STATE_CRCL,
          SWD_HDLC_SND_STATE_CRCH,
          SWD_HDLC_SND_STATE_FLAG,
          SWD_HDLC_SND_STATE_FILL,
          SWD_HDLC_SND_STATE_END
      };

unsigned char ucSWD_HDLC0_SndState;  // Current state of transmit state machine

/******************** Buffer for CRC generation **************************/
typedef union {
               struct {
                        unsigned char low;
                        unsigned char high;
                       } byte;
               unsigned short word;
              } TCRC;

TCRC CRC_HDLC0_Snd;   // CRC buffer for HDLC0 transmit side
TCRC CRC_HDLC0_Rcv;   // CRC buffer for HDLC0 receive side

/************************** Variable definition ***************************/
unsigned char * pucSWD_HDLC0_SndInput;// pointer to transmit input user array
                                      // -> copy of function call parameter SWD_HDLC0_SndIn
unsigned short usSWD_HDLC0_SndLength; // length of transmit input array of user
                                      // -> copy of function call parameter SWD_HDLC0_SndIn
unsigned short usSWD_HDLC0_SndIndex;  // index to current address inside of array
                                      // at pucSWD_HDLC0_SndInput
unsigned char ucSWD_HDLC0_DummyRead;  // Variable for dummy read process

unsigned char ucHDLC0_RcvOut[BUFFER_SIZE]; // Output buffer array for received HDLC data

unsigned long ulSWD_HDLC0_RcvIndex;  // Index for writing into output buufer array

// counter for errors or events
unsigned long ulSWD_HDLC0_CntRcvCRCError;
unsigned long ulSWD_HDLC0_CntRcvBufferOverRun;
unsigned long ulSWD_HDLC0_CntRcvCRCFrameOK;
unsigned long ulSWD_HDLC0_CntRcvAbortError;
unsigned long ulSWD_HDLC0_CntRcvFrameShort;
unsigned long ulSWD_HDLC0_CntRcvG0RI_NotEmpty;
unsigned long ulSWD_HDLC0_CntSndFrameStarted;
unsigned long ulSWD_HDLC0_CntSndFrameOK;
unsigned long ulSWD_HDLC0_CntSndG0TO_NotReady;
unsigned long ulSWD_HDLC0_CntSndG0TB_NotReady;

/*****************************************************************************/
/************************ LOCAL HEADER END ***********************************/
/*****************************************************************************/
```

```
/*****************************************************************************/
/*                                                                           */
/* Subroutine:  SWD_HDLC0_Basetimer_Init                                     */
/* Purpose:     Initialisation of Basetimer0                                 */
/*              Used as clock for Intelligent I/O Group0                     */
/* Parameter:   No                                                           */
/* Return:      No                                                           */
/*                                                                           */
/*****************************************************************************/
void SWD_HDLC0_Basetimer_Init(void)
{
   unsigned short usValue;

   /* Group2 Base Timer Control Register0 */
   // This is needed because of relationship of BTSR register and II/O group2 basetimer
   G2BCR0 |= 0x7F; // XXXX XXXX
                   // |||| ||++- Count source select bit
                   // |||| ||    00: Clock stop
                   // |||| ||    01: fpll
                   // |||| ||    10: Inhibited
                   // |||| ||    11: f1
                   // |+++ ++--- Count source division factor
                   // |          00000: Division by 2
                   // |          00001: Division by 4
                   // |          00010: Division by 6
                   // |          xxxxx:
                   // |          11111: No division
                   // +--------- Base timer Interrupt select bit
                   //            0: Bit 15 overflow
                   //            1: Bit 14 overflow

   /* Base Timer Start Register */
   // Reset basetimer0 start flag
   BTSR &= ~0x01;

   /* Group0 Base Timer Control Register0 */
   // Count source f1 selected, no division by prescaler selected
   G0BCR0 = 0x7F;  // XXXX XXXX
                   // |||| ||++- Count source select bit
                   // |||| ||    00: Clock stop
                   // |||| ||    01: fpll
                   // |||| ||    10: Inhibited
                   // |||| ||    11: f1
                   // |+++ ++--- Count source division factor
                   // |          00000: Division by 2
                   // |          00001: Division by 4
                   // |          00010: Division by 6
                   // |          xxxxx:
                   // |          11111: No division
                   // +--------- Base timer Interrupt select bit
                   //            0: Bit 15 overflow
                   //            1: Bit 14 overflow

   /* Group0 Base Timer Control Register1 */
   // Set basetimer reset cause to channel 0  value match
   // Set basetimner count start
   G0BCR1 = 0x02;  // XXXX XXXX
                   // |||| |||+- Base Timer reset Cause Select Bit 0
                   // |||| |||   0: Synchronizes the base timer reset with n-1 without reseting timer
                   // |||| |||   1: Synchronizes the base timer reset with n-1 with reseting timer
                   // |||| ||+-- Base Timer reset Cause Select Bit 1
                   // |||| ||    0: Does not reset the base timer when it matches WG register ch0
                   // |||| ||    1: Reset the base timer when it matches WG register ch0
                   // |||| |+--- Base Timer reset Cause Select Bit 2
                   // |||| |     0: Does not reset the base timer when input tp the INT pin is L level
                   // |||| |     1: Reset the base timer when input to the INT pin is L level
                   // |||+------ Base Timer start bit
                   // |||        0: Base timer reset
                   // |||        1: Base timer count start
                   // |++------- Up/Down mode control
                   // |          00: Up mode
                   // |          01: Up/Down mode
                   // |          10: Inhibited
                   // |          11: Inhibited
                   // +--------- Base timer Interrupt select bit
```

```
                //            0: 16 bit timer
                //            1: 32 bit timer

/* Group0 Waveform generation control register0 */
G0POCR0 = 0x00; // XXXX XXXX
                // |||| |+++- Operation mode select bit
                // |||| |     000: Single PWM mode
                // |||| |     001: S-R PWM mode
                // |||| |     010: Phase delayed PWM mode
                // |||| |     011: Inhibited
                // |||| |     100: Inhibited
                // |||| |     101: Inhibited
                // |||| |     110: Inhibited
                // |||| |     111: Assigns communication output to a port
                // |||| +---- Must be set to zero
                // |||+------ Output initial value select bit
                // |||         0: Output 0 as the initial value
                // |||         1: Output 1 as the initial value
                // ||+------- Reload timing select bit
                // ||          0: Reloads a new count when CPU writes the count
                // ||          1: Reloads a new count when the base timer 0 is reset
                // |+-------- Must be set to zero
                // +--------- Inverted output function select bit
                //            0: Output is not inverted
                //            1: Output is inverted

/* Group0 Waveform generation control register1 */
G0POCR1 = 0x00; // XXXX XXXX
                // |||| |+++- Operation mode select bit
                // |||| |     000: Single PWM mode
                // |||| |     001: S-R PWM mode
                // |||| |     010: Phase delayed PWM mode
                // |||| |     011: Inhibited
                // |||| |     100: Inhibited
                // |||| |     101: Inhibited
                // |||| |     110: Inhibited
                // |||| |     111: Assigns communication output to a port
                // |||| +---- Must be set to zero
                // |||+------ Output initial value select bit
                // |||         0: Output 0 as the initial value
                // |||         1: Output 1 as the initial value
                // ||+------- Reload timing select bit
                // ||          0: Reloads a new count when CPU writes the count
                // ||          1: Reloads a new count when the base timer 0 is reset
                // |+-------- Must be set to zero
                // +--------- Inverted output function select bit
                //            0: Output is not inverted
                //            1: Output is inverted

// Note: Register G0PO0 have to be set to higher value than G0PO1
usValue = 0x10;   // around 1.666 MHz

/* Group0 waveform generation register0  (for transmitting purpose) */
G0PO0 = usValue;

/* Group0 waveform generation register1  (for receiving purpose) */
G0PO1 = usValue/2;

/* Group0 function enable register */
// Enable channel 0 and 1
G0FE = 0x03;    // XXXX XXXX
                // |||| |||+- Channel 0 enable bit
                // |||| |||   0: Disable
                // |||| |||   1: Enable
                // |||| ||+-- Channel 1 enable bit
                // |||| ||    0: Disable
                // |||| ||    1: Enable
                // |||| |+--- Channel 2 enable bit
                // |||| |     0: Disable
                // |||| |     1: Enable
                // |||| +---- Channel 3 enable bit
                // ||||       0: Disable
                // ||||       1: Enable
                // |||+------ Channel 4 enable bit
                // |||        0: Disable
```

```
                    // |||        1: Enable
                    // ||+------- Channel 5 enable bit
                    // ||         0: Disable
                    // ||         1: Enable
                    // |+-------- Channel 6 enable bit
                    // |          0: Disable
                    // |          1: Enable
                    // +--------- Channel 7 enable bit
                    //            0: Disable
                    //            1: Enable

   /* Base Timer Start Register */
   // Start basetimer Group0
   BTSR |= 0x01;    // XXXX XXXX
                    // |||| |||+- Group0 base timer start bit
                    // |||| |||   0: Base timer reset
                    // |||| |||   1: Base timer count start
                    // |||| ||+-- Group1 base timer start bit
                    // |||| ||    0: Base timer reset
                    // |||| ||    1: Base timer count start
                    // |||| |+--- Group2 base timer start bit
                    // |||| |     0: Base timer reset
                    // |||| |     1: Base timer count start
                    // |||| +---- Group3 base timer start bit
                    // ||||       0: Base timer reset
                    // ||||       1: Base timer count start
                    // ++++------ Nothing is assigned
}

/************************************************************************/
/*                                                                      */
/* Subroutine:  SWD_HDLC0_Init                                          */
/* Purpose:     Initialization of HDLC0 block of Intelligent I/O group 0*/
/*              General init of receive and transmit part               */
/* Parameter:   No                                                      */
/* Return:      No                                                      */
/*                                                                      */
/************************************************************************/
void SWD_HDLC0_Init (void)
{
   SWD_HDLC0_Basetimer_Init();

   /* Set SIO special communication mode */
   // Set HDLC special communication mode
   G0MR = 0x03;     // XXXX XXXX
                    // |||| ||++- special communication mode
                    // |||| ||    BRG count source select bit
                    // |||| ||    00: output compare
                    // |||| ||    01: SIO
                    // |||| ||    10: BEAN
                    // |||| ||    11: HDLC
                    // |||| |+--- CKDIR
                    // |||| |     0: internal clock
                    // |||| |     1: external clock
                    // |||| +---- STPS (stop bits in UART mode)
                    // ||||       0: 1 stop bit
                    // ||||       1: 2 stop bits
                    // |||+------ PRY (UART parity)
                    // |||        0: Odd parity
                    // |||        1: Even parity
                    // ||+------- PRYE (UART parity enable bit)
                    // ||         0: disable
                    // ||         1: enable
                    // |+-------- UFORM (transfer direction select bit)
                    // |          0: LSB first
                    // |          1: MSB first
                    // +--------- IRS (transmit IR cause select)
                    //            0: transmit buffer empty
                    //            1: transmit shift register empty

   /* Set communication control register */
   // Disable transmit and receice block
   G0CR &= ~0x30;   // XXXX XXXX
                    // |||| |||+- TI (Transmit Buffer empty Flag)
                    // |||| |||   0: data in transmit buffer register
```

```
                    // |||| |||   1: no data present in transmit buffer register
                    // |||| ||+-- TXEPT (Transmit register empty flag)
                    // |||| ||    0: data present in transmit register
                    // |||| ||    1: no data present in transmit register
                    // |||| |+--- RI (Receive complete flag)
                    // |||| |     0: no data present in receive buffer register
                    // |||| |     1: data present in receive buffer register
                    // |||| +---- nothing is assigned, when write set to 0
                    // |||+------ TE (Transmit enable bit)
                    // |||        0: Transmission disabled
                    // |||        1: Transmission enable
                    // ||+------- RE (Receive enable bit)
                    // ||         0: Reception disabled
                    // ||         1: Reception enabled
                    // |+-------- IPOL (Receive input polarity reverse select bit)
                    // |          0: no reverse
                    // |          1: reverse
                    // +--------- OPOL (Transmit output polarity reverse select bit)
                    //            0: no reverse
                    //            1: reverse

  /* Set communication control register */
  // Enable receice block
  G0CR |= 0x20;     // XXXX XXXX
                    // |||| |||+- TI (Transmit Buffer empty Flag)
                    // |||| |||   0: data in transmit buffer register
                    // |||| |||   1: no data present in transmit buffer register
                    // |||| ||+-- TXEPT (Transmit register empty flag)
                    // |||| ||    0: data present in transmit register
                    // |||| ||    1: no data present in transmit register
                    // |||| |+--- RI (Receive complete flag)
                    // |||| |     0: no data present in receive buffer register
                    // |||| |     1: data present in receive buffer register
                    // |||| +---- nothing is assigned, when write set to 0
                    // |||+------ TE (Transmit enable bit)
                    // |||        0: Transmission disabled
                    // |||        1: Transmission enable
                    // ||+------- RE (Receive enable bit)
                    // ||         0: Reception disabled
                    // ||         1: Reception enabled
                    // |+-------- IPOL (Receive input polarity reverse select bit)
                    // |          0: no reverse
                    // |          1: reverse
                    // +--------- OPOL (Transmit output polarity reverse select bit)
                    //            0: no reverse
                    //            1: reverse

  /* Set function expand receive control register */
  // Enable flag detection, bit enstuffing, receive CRC and switch on receive switch
  G0ERC = 0xB8;     // XXXX XXXX
                    // |||| |||+- CMP0E (compare 0 trigger enable) for Abort detection
                    // |||| |||   0: disable
                    // |||| |||   1: enable
                    // |||| ||+-- CMP1E (comapre 1 enable)
                    // |||| ||    0: disable
                    // |||| ||    1: enable
                    // |||| |+--- CMP2E (compare 2 enable)
                    // |||| |     0: disable
                    // |||| |     1: enable
                    // |||| +---- CMP3E (compare 3 enable) for Flag detection
                    // ||||       0: disable
                    // ||||       1: enable
                    // |||+------ RCRCE (receive CRC enable)
                    // |||        0: disable
                    // |||        1: enable
                    // ||+------- RSHTE (receive shift switch)
                    // ||         0: switched off
                    // ||         1: switched on
                    // |+-------- RBSF0 (bit enstuffing 1 deletion)
                    // |          0: disable
                    // |          1: enable
                    // +--------- RBSF1 (bit enstuffing 0 deletion)
                    //            0: disable
                    //            1: enable
```

```
/* Set compare register values */
G0CMP0 = SWD_HDLC0_ABORT;       // Group 0 data compare register 0 set to 0xFE
G0MSK0 = SWD_HDLC0_ABORT_MASK;  // Group 0 data mask register 0 set to 0x01
G0CMP3 = SWD_HDLC0_FLAG;        // Group 0 data compare register 3 set to 0x7E

/* Set function expand transmit control register */
G0ETC = 0x00;    // XXXX XXXX
                 // |||| +---- SOF (SOF transmit request bit)
                 // ||||       0: No SOF transmit request
                 // ||||       1: SOF transmit request
                 // |||+------ TCRCE (transmit CRC enable)
                 // |||        0: disable
                 // |||        1: enable
                 // ||+------- ABTE (arbitration enable bit)
                 // ||         0: disable
                 // ||         1: enable
                 // |+-------- TBSF0 (bit stuffing 1 insertion)
                 // |          0: disable
                 // |          1: enable
                 // +--------- TBSF1 (bit stuffing 0 insertion)
                 //            0: disable
                 //            1: enable

/* Set interrupt priority level for Intelligent I/O group0 to zero */
IIO0IC &= ~0x07;    // For Intelligent I/O group0 receive
IIO1IC &= ~0x07;    // For Intelligent I/O group0 transmit
IIO4IC &= ~0x07;    // For compare trigger

/* Set function expand mode register */
// Enable auto CRC init and set 0xFFFF as init value
// Set CRC polynom, select parallel output register
G0EMR = 0xE6;    // XXXX XXXX
                 // |||| |||+- SMODE (used for BEAN)
                 // |||| |||   0: Normal mode
                 // |||| |||   1: Resynchronous mode
                 // |||| ||+-- CRCV (CRC initialize value bit)
                 // |||| ||    0: initialize 0x0000
                 // |||| ||    1: initialize 0xFFFF
                 // |||| |+--- ACRC (CRC initialization select bit)
                 // |||| |     0: no auto init
                 // |||| |     1: auto init: bit stuffing and CRC are initialized by cmp3 trigger
                 // |||| +---- BSINT: enable bit stuffing error interrupt select bit
                 // ||||       0: disable
                 // ||||       1: enable
                 // |||+------ RXSL (receive root select bit)
                 // |||        0: RxD (BEAN)
                 // |||        1: Receive Input Buffer (HDLC)
                 // ||+------- TXSL (transmit destination select bit)
                 // ||         0: TxD (BEAN)
                 // ||         1: Transmit Output Register (HDLC)
                 // ++-------- CRC0, CRC1 (CRC polynom selection)
                 //            00: x8+x4+x+1 (BEAN)
                 //            01: invalid
                 //            10: x16+x15+x2+1
                 //            11: x16+x12+x5+1 (HDLC)

/* Set dummy data to receive input buffer of II/O group0 */
G0RI = 0xFF;

// Wait for a basetimer clock cycle, before input select is switched to register input
while ( G0BT >= 0x0010);

/* Set function expand mode register */
// Select parallel input register
G0EMR = 0xF6;    // XXXX XXXX
                 // |||| |||+- SMODE (used for BEAN)
                 // |||| |||   0: Normal mode
                 // |||| |||   1: Resynchronous mode
                 // |||| ||+-- CRCV (CRC initialize bit)
                 // |||| ||    0: initialize 0x0000
                 // |||| ||    1: initialize 0xFFFF
                 // |||| |+--- ACLR (auto clear bit)
                 // |||| |     0: no auto clear
                 // |||| |     1: auto clear: bit stuffing and CRC are initialized by cmp3 trigger
                 // |||| +---- BS_INT: enable bit stuffing error trigger
```

```
                    // ||||       0: disable
                    // ||||       1: enable
                    // |||+------ RXSL (receive root select bit)
                    // |||        0: RxD (BEAN)
                    // |||        1: Receive Input Buffer (HDLC)
                    // ||+------- TXSL (transmit destination select bit)
                    // ||         0: TxD (BEAN)
                    // ||         1: Transmit Output Register (HDLC)
                    // ++-------- CRC0, CRC1 (CRC polynom selection)
                    //            00: x8+x4+x+1 (BEAN)
                    //            01: x12+x11+x3+x2+1
                    //            10: x16+x15+x2+1
                    //            11: x16+x12+x5+1 (HDLC)

   // Initialization of all error and event counter
   ulSWD_HDLC0_CntRcvCRCError = ZERO;
   ulSWD_HDLC0_CntRcvBufferOverRun = ZERO;
   ulSWD_HDLC0_CntRcvCRCFrameOK = ZERO;
   ulSWD_HDLC0_CntRcvAbortError = ZERO;
   ulSWD_HDLC0_CntRcvFrameShort = ZERO;
   ulSWD_HDLC0_CntRcvG0RI_NotEmpty = ZERO;
   ulSWD_HDLC0_CntSndFrameStarted = ZERO;
   ulSWD_HDLC0_CntSndFrameOK = ZERO;
   ulSWD_HDLC0_CntSndG0TO_NotReady = ZERO;
   ulSWD_HDLC0_CntSndG0TB_NotReady = ZERO;

   msgSWD_HDLC0.TransmitStart = SWD_HDLC0_NO; // Set message, that transmission is finished
   msgSWD_HDLC0.ReceiveFlagDetected = SWD_HDLC0_NO; // Set message that no flag has been detected

   ulSWD_HDLC0_RcvIndex = ZERO;  // Init receive HDLC data index
}

/*****************************************************************************/
/*                                                                           */
/* Subroutine:  SWD_HDLC0_RcvPoll                                            */
/* Purpose:     Routine, which have to be polled in application software      */
/*              This routine takes care about the needed action, if the      */
/*              received data is equal to HDLC specific pattern, e.g.        */
/*              StartFlag or Abort.                                          */
/*                                                                           */
/* Parameter:   No                                                           */
/* Return:      No                                                           */
/*                                                                           */
/*****************************************************************************/
void SWD_HDLC0_RcvPoll(void)
{
 if(G0IRF & 0x80)        // Has a start/end flag been detection by Compare Register?
                         // Therefore check compare match register 3 flag at
                         // special communication interrupt detect register
 {
   ucSWD_HDLC0_DummyRead = G0RB;     // Dummy read out of receive output buffer,
   IIO0IR &= ~0x20;                  // Reset interrupt request for receive output buffer full

   G0IRF &= ~0x80;    // Reset start or end flag detection interrupt request

   /* set function expand receive control register */
   // Enable Abort and flag detection, bit enstuffing 0 enable, receive shift on, CRC enable
   G0ERC = 0xB9;

   // If received number of bytes not enough for a complete frame
   if(ulSWD_HDLC0_RcvIndex < SWD_HDLC0_FRAME_MIN)
   {
     ulSWD_HDLC0_RcvIndex = ZERO;     // Reset receive counter
     ulSWD_HDLC0_CntRcvFrameShort++;  // Increment receive frame to short error counter
     msgSWD_HDLC0.ReceiveFlagDetected = SWD_HDLC0_YES; // Set message that a flag has been detected
   }

   else if(msgSWD_HDLC0.ReceiveFlagDetected == SWD_HDLC0_NO)
   {
     msgSWD_HDLC0.ReceiveFlagDetected = SWD_HDLC0_YES;   // Set message that a flag has been detected
   }

   else if(msgSWD_HDLC0.ReceiveFlagDetected == SWD_HDLC0_YES)
   {
     CRC_HDLC0_Rcv.word = G0RCRC;   // Read out crc result
```

```
    if(CRC_HDLC0_Rcv.word != SWD_HDLC0_GOOD_CRC)  // A wrong CRC has been generated
    {
      ulSWD_HDLC0_RcvIndex = ZERO;    // Reset receive index counter
      ulSWD_HDLC0_CntRcvCRCError++;   // Increment receive CRC error counter
    }

    else  // CRC has been generated correct
    {
      // ----------------------------------------
      // -------- Handle for correct CRC ---------
      // ----------------------------------------
      // Please add your code here for received HDLC frame here
      ulSWD_HDLC0_RcvIndex = ZERO;     // Reset receive index counter
      ulSWD_HDLC0_CntRcvCRCFrameOK++;  // Increment receive CRC OK counter
    }
   } // ENDOF else if(msgSWD_HDLC0.ReceiveFlagDetected == SWD_HDLC0_YES)
 } //ENDOF if(G0IRF & 0x80)

 else if(G0IRF & 0x10) // Has a abort flag been detection by Compare Register?
                       // Therefore check compare match register 0 flag at
                       // special communication interrupt detect register
 {
    ucSWD_HDLC0_DummyRead = G0RB;  // Dummy read out of receive output buffer,
    IIO0IR &= ~0x20;               // Reset interrupt request for receive output buffer full

    G0IRF &= ~0x10;  // Reset Abort Flag detection interrupt request

    /* set function expand receive control register */
    // Disable Abort and enable flag detection, bit enstuffing 0 enable, receive shift on, CRC enable
    G0ERC = 0xB8;

    msgSWD_HDLC0.ReceiveFlagDetected = SWD_HDLC0_NO; // Set message that no flag has been detected
    ulSWD_HDLC0_RcvIndex = ZERO;           // Reset receive index counter
    ulSWD_HDLC0_CntRcvAbortError++;        // Increment Abort flag counter
 } //ENDOF else if(G0IRF & 0x10)
}

/*****************************************************************************/
/*                                                                           */
/* Subroutine:  SWD_HDLC0_RcvIn                                              */
/* Purpose:     Routine, which have to be polled in application software      */
/*              Within this routine received HDLC data will be handled,      */
/*              and routed to the received input buffer register.            */
/*                                                                           */
/* Parameter:   No                                                           */
/* Return:      No                                                           */
/*                                                                           */
/*****************************************************************************/
void SWD_HDLC0_RcvIn(unsigned char * pucRcvIn, unsigned char ucSlot)
{
  if (IIO0IR & 0x10)   // Interrupt requested flag is set for G0RI register?
  {
    IIO0IR &= ~0x10;   // Reset interrupt request flag
    G0RI = * (pucRcvIn + ucSlot); // Write received data byte in HDLC block
                                                 // receive input buffer
  }
  else
      ulSWD_HDLC0_CntRcvG0RI_NotEmpty++; // Increment G0RI not empty counter
}

/*****************************************************************************/
/*                                                                           */
/* Subroutine:  SWD_HDLC0_RcvOut                                             */
/* Purpose:     Routine, which have to be polled in application software      */
/*              Within this routine received HDLC data will be handled,      */
/*              and routed to the received output buffer array.              */
/*                                                                           */
/* Parameter:   No                                                           */
/* Return:      No                                                           */
/*                                                                           */
/*****************************************************************************/
void SWD_HDLC0_RcvOut(void)
{
  if (IIO0IR & 0x20)   // Interrupt has been requested by G0RB register?
```

```
  {
    IIO0IR &= ~0x20;   // Reset interrupt request flag
    ucHDLC0_RcvOut[ulSWD_HDLC0_RcvIndex] = G0RB; // Read receive output
                                                 // buffer value and store
                                                 // in buffer
    ulSWD_HDLC0_RcvIndex++;  // Increment received bytes index (for current frame)
    if (ulSWD_HDLC0_RcvIndex >= BUFFER_SIZE-1) // Received frame is too long to store in buffer?
    {
       ulSWD_HDLC0_RcvIndex = BUFFER_SIZE-1;   // Set index to max value
       ulSWD_HDLC0_CntRcvBufferOverRun++;      // Increment buffer overrun counter
    }
  }
}

/*****************************************************************************/
/*                                                                           */
/* Subroutine:  SWD_HDLC0_SndIn                                              */
/* Purpose:     This function starts the start/end flag, stuffing and CRC    */
/*              procedure. To do so, a pointer to an unprocessed frame       */
/*              and the number of bytes of the frame are needed as input     */
/*              parameters for this function.                                */
/*              If this function is called with a new frame as parameter,    */
/*              before the previous one has been processed complete, an      */
/*              error flag will be returned.                                 */
/*              If start procedure was successful, an OK flag will be        */
/*              returned.                                                    */
/*                                                                           */
/* Parameter:#1 unsigned char * pSndInput - pointer to the buffer where      */
/*              the unprocessed frame is stored                              */
/*           #2 unsigned short usLength_SndInput - length of the frame       */
/* Return:      unsigned char - Error flag, if function is called twice      */
/*                              OK flag, if process has been started         */
/*                                                                           */
/*****************************************************************************/
unsigned char SWD_HDLC0_SndIn (unsigned char * pucSndInput,
                               unsigned short usLength_SndInput)
{
   // To check whether Start routine have been entered twice, before previous frame
   // has been processed complete
   if (msgSWD_HDLC0.TransmitStart == SWD_HDLC0_YES)
   {
     return(SWD_HDLC0_ERROR);   // Left function with error feedback
   }

   msgSWD_HDLC0.TransmitStart = SWD_HDLC0_YES;  // Set message, that transmission have been started now

   /* Set communication control register */
   // Enable transmit part
   G0CR |= 0x10;      // XXXX XXXX
                      // |||| |||+- TI (Transmit Buffer empty Flag)
                      // |||| |||   0: data in transmit buffer register
                      // |||| |||   1: no data present in transmit buffer register
                      // |||| ||+-- TXEPT (Transmit register empty flag)
                      // |||| ||    0: data present in transmit register
                      // |||| ||    1: no data present in transmit register
                      // |||| |+--- RI (Receive complete flag)
                      // |||| |     0: no data present in receive buffer register
                      // |||| |     1: data present in receive buffer register
                      // |||| +---- nothing is assigned, when write set to 0
                      // |||+------ TE (Transmit enable bit)
                      // |||        0: Transmission disabled
                      // |||        1: Transmission enable
                      // ||+------- RE (Receive enable bit)
                      // ||         0: Reception disabled
                      // ||         1: Reception enabled
                      // |+-------- IPOL (Receive input polarity reverse select bit)
                      // |          0: no reverse
                      // |          1: reverse
                      // +--------- OPOL (Transmit output polarity reverse select bit)
                      //            0: no reverse
                      //            1: reverse

   pucSWD_HDLC0_SndInput = pucSndInput;  // Set pointer to unprocessed frame
                                         // buffer based on function call parameter
   ucSWD_HDLC0_SndState = SWD_HDLC_SND_STATE_DATA; // Initialize transmit state machine
```

```
   usSWD_HDLC0_SndIndex = ZERO;                    // Initialize send index with 0
   usSWD_HDLC0_SndLength = usLength_SndInput; // Initialize transmit length based
                                              // on function call parameter, minus
                                              // one becaus following index
                                              // usSWD_HDLC0_SndIndex starts at zero

   G0TB = SWD_HDLC0_FLAG;    // Transfer start flag to transmit input buffer,
                             // which actually starts the transfer process

   CRC_HDLC0_Snd.word = SWD_HDLC0_CRC_INIT; // Init of backup CRC, respectively Standard CRC

   ulSWD_HDLC0_CntSndFrameStarted++; // Increment counter of transmitted frames (just started)

   return(SWD_HDLC0_OK);
}

/*****************************************************************************/
/*                                                                           */
/* Subroutine:  SWD_HDLC0_SndOut                                             */
/* Purpose:     Routine, which have to be polled in application software.    */
/*              Within this routine transmit HDLC data will be handled,      */
/*              and routed to the specific address, assigned by the          */
/*              parameter.                                                   */
/*                                                                           */
/* Parameter:#1 unsigned char *pucSndOut - pointer to the GCI frame          */
/*           #2 unsigned char ucSlot - index parameter for GCI frame         */
/* Return:      No                                                           */
/*                                                                           */
/*****************************************************************************/
void SWD_HDLC0_SndOut(unsigned char * pucSndOut, unsigned char ucSlot)
{
  if (IIO1IR & 0x10)   // Interrupt requested flag is set for G0TO register
  {
    IIO1IR &= ~0x10; // Reset interrupt request flag
    * (pucSndOut + ucSlot) = G0TO; // Transfer HDLC transmit output buffer
                                   // to GCI frame
  }
  else // If no data transfer is currently performed
  {
    * (pucSndOut + ucSlot) = SWD_HDLC0_PAUSE; // Transfer HDLC pause data to GCI buffer
    ulSWD_HDLC0_CntSndG0TO_NotReady++;  // Increment G0TO not ready counter
  }
}

/*****************************************************************************/
/*                                                                           */
/* Subroutine:  SWD_HDLC0_SndPoll                                            */
/* Purpose:     Routine, which have to be polled in application software.    */
/*              The statemachine takes care about stuffing, CRC, end flag    */
/*              generation                                                   */
/*                                                                           */
/* Parameter:   No                                                           */
/* Return:      No                                                           */
/*                                                                           */
/*****************************************************************************/
void SWD_HDLC0_SndPoll(void)
{
  unsigned char ucDummy;

  if(IIO1IR & 0x20)   // Confirm interrupt request of transmit input buffer (G0TB)
  {
    IIO1IR &= ~0x20;  // Reset interrupt request flag

    if (msgSWD_HDLC0.TransmitStart == SWD_HDLC0_YES)  // Check message, whether transmission has been
                                                      // started
    {
      switch (ucSWD_HDLC0_SndState)         // Switch by transmit state machine
      {
        case SWD_HDLC_SND_STATE_DATA:
            G0ETC |= 0x80;                                      // Enable bit stuffing
            G0TB = *(pucSWD_HDLC0_SndInput+usSWD_HDLC0_SndIndex); // Transfer byte to transmit input
                                                                  // register
            CRCD = CRC_HDLC0_Snd.word;                          // Use backup value as new init value
            CRCIN = *(pucSWD_HDLC0_SndInput+usSWD_HDLC0_SndIndex); // Generate new CRC
            asm("NOP");                                         // wait two cycles
```

```
            asm("NOP");
            CRC_HDLC0_Snd.word = CRCD;                          // Backup CRC again
            usSWD_HDLC0_SndIndex++;                             // Increment index counter
            if(usSWD_HDLC0_SndIndex >= usSWD_HDLC0_SndLength) // Compare index counter with actually
                                                                // number of bytes, which is reached.
            {                                                  // (Actually one byte before end of frame)
              ucSWD_HDLC0_SndState = SWD_HDLC_SND_STATE_CRCL; // If end of frame is reached, change to
                                                                // new state machine state
            }
            break;

        case SWD_HDLC_SND_STATE_CRCL:
            CRC_HDLC0_Snd.word = ~CRC_HDLC0_Snd.word;            // Invert the complete CRC word
            G0TB = CRC_HDLC0_Snd.byte.low;     // Transfer the low byte of the CRC word to transmit input
                                               // register
            ucSWD_HDLC0_SndState = SWD_HDLC_SND_STATE_CRCH;  // Change to new state machine state
            break;

        case SWD_HDLC_SND_STATE_CRCH:
            G0TB = CRC_HDLC0_Snd.byte.high;   // Transfer the low byte of the CRC word to transmit input
                                              // register
            ucSWD_HDLC0_SndState = SWD_HDLC_SND_STATE_FLAG;   // Change to new state machine state
            break;

        case SWD_HDLC_SND_STATE_FLAG:
            G0ETC &= ~0x80;                                 // Disable bit stuffing
            G0TB = SWD_HDLC0_FLAG;                         // Transfer end flag to transmit input register
            ucSWD_HDLC0_SndState = SWD_HDLC_SND_STATE_FILL; // Change to new state machine state
            break;

        case SWD_HDLC_SND_STATE_FILL:
            G0MR |= 0x80;              // Change interrupt request trigger to input transmit is completed
            G0TB = SWD_HDLC0_FILL;   // Transfer second fill byte to transmit input register
            ucSWD_HDLC0_SndState = SWD_HDLC_SND_STATE_END;  // Change to new state machine state
            break;

        case SWD_HDLC_SND_STATE_END:
            G0MR &= ~0x80;       // Change interrupt request trigger to transmit input buffer empty (G0TB)
            G0CR &= ~0x10;        // Disable transmit block
            IIO1IR &= ~0x30;     // Reset interrupt request flag for input and output transmit register
            ucDummy = G0TO;       // Dummy read is needed
            msgSWD_HDLC0.TransmitStart = SWD_HDLC0_NO; // Set message, that transmission is finished
            ulSWD_HDLC0_CntSndFrameOK++;
            break;

        default:
            // Wrong state!!!!
            break;

       } // ENDOF switch
      } // ENDOF if(msgSWD_HDLC0.TransmitStart == SWD_HDLC0_YES)
    } // ENDOF if(IIO1IR & 0x20)
    else
       ulSWD_HDLC0_CntSndG0TB_NotReady++;
}

/********************************** E O F *******************************/
```

5.6 SWD_HDLC1.h

```
/*****************************************************************************/
/*****************************************************************************/
/***************** Renesas Technology Europe GmbH ****************************/
/*****************************************************************************/
/*****************************************************************************/
/*  DISCLAIMER                                                               */
/*  We (Renesas Technology Europe GmbH) do not warrant that the Software     */
/*  is free from claims by a third party of copyright, patent, trademark,    */
/*  trade secret or any other intellectual property infringement.            */
/*                                                                           */
/*  Under no circumstances are we liable for any of the following:           */
/*                                                                           */
/*  1. third-party claims against you for losses or damages;                 */
```

```
/*  2. loss of, or damage to, your records or data; or                    */
/*  3. economic consequential damages (including lost profits or          */
/*     savings) or incidental damages, even if we are informed of         */
/*     their possibility.                                                 */
/*                                                                        */
/*  We do not warrant uninterrupted or error free operation of the        */
/*  Software. We have no obligation to provide service, defect            */
/*  correction, or any maintenance for the Software. We have no           */
/*  obligation to supply any Software updates or enhancements to you      */
/*  even if such are or later become available.                           */
/*                                                                        */
/*  IF YOU DOWNLOAD OR USE THIS SOFTWARE YOU AGREE TO THESE TERMS.        */
/*                                                                        */
/*  THERE ARE NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING THE            */
/*  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A               */
/*  PARTICULAR PURPOSE.                                                   */
/************************************************************************/
/************************************************************************/
/*                                                                      */
/*  Name:        SWD_HDLC1.h                                            */
/*  Part of:     M32C_HDLC_SoftwareDriver                               */
/*  Description: global definitions and declarations for HDLC function  */
/*  Date  :      23.04.2003                                             */
/*  Author:      BWE @ Renesas Technology Europe GmbH                   */
/*  Change:      (Date)  (Author)  (Description)                        */
/*                                                                      */
/************************************************************************/

/************************************************************************/
/*************************** GLOBAL HEADER PART *************************/
/************************************************************************/
/*************************** General makros *****************************/
#define SWD_HDLC1_OK     0x00
#define SWD_HDLC1_ERROR  0x01

#define SWD_HDLC1_NO 0
#define SWD_HDLC1_YES 1

/*************************** Function prototypes ************************/
// HDLC1 initialization
EXTERN void SWD_HDLC1_Init(void);
// Function which have to be polled by user to start transmission
EXTERN unsigned char SWD_HDLC1_SndIn(unsigned char *, unsigned short);
// Function which have to be polled by user for transmission output
EXTERN void SWD_HDLC1_SndOut(unsigned char *, unsigned char);
// Function which have to be polled by user for transmission
EXTERN void SWD_HDLC1_SndPoll(void);
// Function which have to be polled by user for reception
EXTERN void SWD_HDLC1_RcvPoll(void);
// Function which have to be polled by user for reception input
EXTERN void SWD_HDLC1_RcvIn(unsigned char *, unsigned char);
// Function which have to be polled by user for reception output
EXTERN void SWD_HDLC1_RcvOut(void);
/*************************** Variable Definition ************************/

/************************************ E O F ******************************/
```

5.7 SWD_HDLC1.c

```
/************************************************************************/
/************************************************************************/
/****************** Renesas Technology Europe GmbH **********************/
/************************************************************************/
/************************************************************************/
/*  DISCLAIMER:                                                         */
/*  We (Renesas Technology Europe GmbH) do not warrant that the Software  */
/*  is free from claims by a third party of copyright, patent, trademark, */
/*  trade secret or any other intellectual property infringement.       */
/*                                                                      */
/*  Under no circumstances are we liable for any of the following:      */
/*                                                                      */
/*  1. third-party claims against you for losses or damages;            */
/*  2. loss of, or damage to, your records or data; or                  */
```

```
/*  3. economic consequential damages (including lost profits or          */
/*     savings) or incidental damages, even if we are informed of         */
/*     their possibility.                                                 */
/*                                                                        */
/*  We do not warrant uninterrupted or error free operation of the        */
/*  Software. We have no obligation to provide service, defect            */
/*  correction, or any maintenance for the Software. We have no           */
/*  obligation to supply any Software updates or enhancements to you      */
/*  even if such are or later become available.                           */
/*                                                                        */
/*  IF YOU DOWNLOAD OR USE THIS SOFTWARE YOU AGREE TO THESE TERMS.        */
/*                                                                        */
/*  THERE ARE NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING THE            */
/*  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A               */
/*  PARTICULAR PURPOSE.                                                   */
/************************************************************************/
/************************************************************************/
/*                                                                      */
/*  Name:        SWD_HDLC1.c                                            */
/*  Part of:     M32C_HDLC_SoftwareDriver                               */
/*  Description: HDLC driver module                                     */
/*  Date  :      23.04.2003                                             */
/*  Author:      BWE @ Renesas Technology Europe GmbH                   */
/*  Change:      (Date)  (Author)  (Description)                        */
/*                                                                      */
/************************************************************************/
#define EXTERN extern
#include   "sfr_3083.h"
#include   "int_3083.h"
#undef EXTERN

#define EXTERN
#include "SWD_HDLC1.h"
#undef EXTERN

/************************************************************************/
/************************ LOCAL HEADER START ****************************/
/************************************************************************/
/* Only one transmit pause mode have to be selected */
//#define MARK_IDLE       // constantly 1 characters as pause signal
#define FLAG_IDLE         // flag characters as pause signal

/************************ Definitions ***********************************/
#define SWD_HDLC1_GOOD_CRC   0x0B9F  // Result for correct CRC reception
                                     // (includes flag)
#define SWD_HDLC1_FLAG       0x7E    // Start / end flag
#define SWD_HDLC1_ABORT      0xFE    // Abort 11111110
#define SWD_HDLC1_ABORT_MASK 0x01    // 1st bit mask for abort
#define SWD_HDLC1_FILL       0xFF    // Data between frames

#ifdef MARK_IDLE
 #define SWD_HDLC1_PAUSE      0xFF   // Ones as pause signal for HDLC traffic
#endif

#ifdef FLAG_IDLE
 #define SWD_HDLC1_PAUSE      0x7E   // flag characters as pause signal
#endif

#define SWD_HDLC1_FRAME_MIN  3      // Set minimum frame size

#define SWD_HDLC1_CRC_INIT 0xFFFF // Init value for CRC generation

#define ZERO 0

#define BUFFER_SIZE 1001    // Receive output buffer size definition

/************************ Function prototypes ***************************/
void SWD_HDLC1_Basetimer_Init(void);    // Initialisation of Basetimer0

/************************ Message structure *****************************/
typedef struct
  {
    unsigned short TransmitStart: 1;       // 1: Process for transmission frame has been started
    unsigned short ReceiveFlagDetected: 1; // 1: Flag Detected
    /*  Place for more messages  */
```

```
  } TSWD_HDLC1_Msg;

TSWD_HDLC1_Msg msgSWD_HDLC1;

/****************** States of transmit state machine *********************/
enum SWD_HDLC_SndEnum
      {
          SWD_HDLC_SND_STATE_DATA,
          SWD_HDLC_SND_STATE_CRCL,
          SWD_HDLC_SND_STATE_CRCH,
          SWD_HDLC_SND_STATE_FLAG,
          SWD_HDLC_SND_STATE_FILL,
          SWD_HDLC_SND_STATE_END
      };

unsigned char ucSWD_HDLC1_SndState;  // Current state of transmit state machine

/******************** Buffer for CRC generation **************************/
typedef union {
               struct {
                       unsigned char low;
                       unsigned char high;
                      } byte;
               unsigned short word;
              } TCRC;

TCRC CRC_HDLC1_Snd;   // CRC buffer for HDLC1 transmit side
TCRC CRC_HDLC1_Rcv;   // CRC buffer for HDLC1 receive side

/************************* Variable definition ***************************/
unsigned char * pucSWD_HDLC1_SndInput;// pointer to transmit input user array
                                      // -> copy of function call parameter SWD_HDLC1_SndIn
unsigned short usSWD_HDLC1_SndLength; // length of transmit input array of user
                                      // -> copy of function call parameter SWD_HDLC1_SndIn
unsigned short usSWD_HDLC1_SndIndex;  // index to current address inside of array
                                      // at pucSWD_HDLC1_SndInput
unsigned char ucSWD_HDLC1_DummyRead;  // Variable for dummy read process

unsigned char ucHDLC1_RcvOut[BUFFER_SIZE]; // Output buffer array for received HDLC data

unsigned long ulSWD_HDLC1_RcvIndex;  // Index for writing into output buufer array

// counter for errors or events
unsigned long ulSWD_HDLC1_CntRcvCRCError;
unsigned long ulSWD_HDLC1_CntRcvBufferOverRun;
unsigned long ulSWD_HDLC1_CntRcvCRCFrameOK;
unsigned long ulSWD_HDLC1_CntRcvAbortError;
unsigned long ulSWD_HDLC1_CntRcvFrameShort;
unsigned long ulSWD_HDLC1_CntRcvG1RI_NotEmpty;
unsigned long ulSWD_HDLC1_CntSndFrameStarted;
unsigned long ulSWD_HDLC1_CntSndFrameOK;
unsigned long ulSWD_HDLC1_CntSndG1TO_NotReady;
unsigned long ulSWD_HDLC1_CntSndG1TB_NotReady;

/*****************************************************************************/
/************************* LOCAL HEADER END **********************************/
/*****************************************************************************/

/*****************************************************************************/
/*                                                                           */
/* Subroutine:  SWD_HDLC1_Basetimer_Init                                     */
/* Purpose:     Initialisation of Basetimer1                                 */
/*              Used as clock for Intelligent I/O Group1                     */
/* Parameter:   No                                                           */
/* Return:      No                                                           */
/*                                                                           */
/*****************************************************************************/
void SWD_HDLC1_Basetimer_Init(void)
{
   unsigned short usValue;

   /* Group2 Base Timer Control Register0 */
   // This is needed because of relationship of BTSR register and II/O group2 basetimer
   G2BCR0 |= 0x7F; // XXXX XXXX
                   // |||| ||++- Count source select bit
```

```
                // ||||  ||    00: Clock stop
                // ||||  ||    01: fpll
                // ||||  ||    10: Inhibited
                // ||||  ||    11: f1
                // |+++ ++---- Count source division factor
                // |           00000: Division by 2
                // |           00001: Division by 4
                // |           00010: Division by 6
                // |           xxxxx:
                // |           11111: No division
                // +---------- Base timer Interrupt select bit
                //             0: Bit 15 overflow
                //             1: Bit 14 overflow

/* Base Timer Start Register */
// Reset basetimer1 start flag
BTSR &= ~0x02;

/* Group1 Base Timer Control Register0 */
// Count source f1 selected, no division by prescaler selected
G1BCR0 = 0x7F;  // XXXX XXXX
                // ||||  ||++- Count source select bit
                // ||||  ||    00: Clock stop
                // ||||  ||    01: fpll
                // ||||  ||    10: Inhibited
                // ||||  ||    11: f1
                // |+++ ++---- Count source division factor
                // |           00000: Division by 2
                // |           00001: Division by 4
                // |           00010: Division by 6
                // |           xxxxx:
                // |           11111: No division
                // +---------- Base timer Interrupt select bit
                //             0: Bit 15 overflow
                //             1: Bit 14 overflow

/* Group1 Base Timer Control Register1 */
// Set basetimer reset cause to channel 0  value match
// Set basetimner count start
G1BCR1 = 0x02;  // XXXX XXXX
                // ||||  |||+- Base Timer reset Cause Select Bit 0
                // ||||  |||   0: Synchronizes the base timer reset with n-1 without reseting timer
                // ||||  |||   1: Synchronizes the base timer reset with n-1 with reseting timer
                // ||||  ||+-- Base Timer reset Cause Select Bit 1
                // ||||  ||    0: Does not reset the base timer when it matches WG register ch0
                // ||||  ||    1: Reset the base timer when it matches WG register ch0
                // ||||  |+--- Base Timer reset Cause Select Bit 2
                // ||||  |     0: Does not reset the base timer when input tp the INT pin is L level
                // ||||  |     1: Reset the base timer when input to the INT pin is L level
                // |||+------ Base Timer start bit
                // |||         0: Base timer reset
                // |||         1: Base timer count start
                // |++------- Up/Down mode control
                // |           00: Up mode
                // |           01: Up/Down mode
                // |           10: Inhibited
                // |           11: Inhibited
                // +---------- Base timer Interrupt select bit
                //             0: 16 bit timer
                //             1: 32 bit timer

/* Group1 Waveform generation control register0 */
G1POCR0 = 0x00; // XXXX XXXX
                // ||||  |+++- Operation mode select bit
                // ||||  |     000: Single PWM mode
                // ||||  |     001: S-R PWM mode
                // ||||  |     010: Phase delayed PWM mode
                // ||||  |     011: Inhibited
                // ||||  |     100: Inhibited
                // ||||  |     101: Inhibited
                // ||||  |     110: Inhibited
                // ||||  |     111: Assigns communication output to a port
                // ||||  +---- Must be set to zero
                // |||+------ Output initial value select bit
                // |||         0: Output 0 as the initial value
```

```
                // |||        1: Output 1 as the initial value
                // ||+------- Reload timing select bit
                // ||         0: Reloads a new count when CPU writes the count
                // ||         1: Reloads a new count when the base timer 1 is reset
                // |+-------- Must be set to zero
                // +--------- Inverted output function select bit
                //            0: Output is not inverted
                //            1: Output is inverted

/* Group1 Waveform generation control register1 */
G1POCR1 = 0x00; // XXXX XXXX
                // |||| |+++- Operation mode select bit
                // |||| |     000: Single PWM mode
                // |||| |     001: S-R PWM mode
                // |||| |     010: Phase delayed PWM mode
                // |||| |     011: Inhibited
                // |||| |     100: Inhibited
                // |||| |     101: Inhibited
                // |||| |     110: Inhibited
                // |||| |     111: Assigns communication output to a port
                // |||| +---- Must be set to zero
                // |||+------ Output initial value select bit
                // |||        0: Output 0 as the initial value
                // |||        1: Output 1 as the initial value
                // ||+------- Reload timing select bit
                // ||         0: Reloads a new count when CPU writes the count
                // ||         1: Reloads a new count when the base timer 1 is reset
                // |+-------- Must be set to zero
                // +--------- Inverted output function select bit
                //            0: Output is not inverted
                //            1: Output is inverted

// Note: Register G1PO0 have to be set to higher value than G1PO1
usValue = 0x10;   // around 1.666 MHz

/* Group1 waveform generation register0  (for transmitting purpose) */
G1PO0 = usValue;

/* Group1 waveform generation register1  (for receiving purpose) */
G1PO1 = usValue/2;

/* Group1 function enable register */
// Enable channel 0 and 1
G1FE = 0x03;    // XXXX XXXX
                // |||| |||+- Channel 0 enable bit
                // |||| |||   0: Disable
                // |||| |||   1: Enable
                // |||| ||+-- Channel 1 enable bit
                // |||| ||    0: Disable
                // |||| ||    1: Enable
                // |||| |+--- Channel 2 enable bit
                // |||| |     0: Disable
                // |||| |     1: Enable
                // |||| +---- Channel 3 enable bit
                // ||||       0: Disable
                // ||||       1: Enable
                // |||+------ Channel 4 enable bit
                // |||        0: Disable
                // |||        1: Enable
                // ||+------- Channel 5 enable bit
                // ||         0: Disable
                // ||         1: Enable
                // |+-------- Channel 6 enable bit
                // |          0: Disable
                // |          1: Enable
                // +--------- Channel 7 enable bit
                //            0: Disable
                //            1: Enable

/* Base Timer Start Register */
// Start basetimer Group0
BTSR |= 0x02;   // XXXX XXXX
                // |||| |||+- Group0 base timer start bit
                // |||| |||   0: Base timer reset
                // |||| |||   1: Base timer count start
```

```
                  // |||| ||+-- Group1 base timer start bit
                  // |||| ||    0: Base timer reset
                  // |||| ||    1: Base timer count start
                  // |||| |+--- Group2 base timer start bit
                  // |||| |     0: Base timer reset
                  // |||| |     1: Base timer count start
                  // |||| +---- Group3 base timer start bit
                  // ||||       0: Base timer reset
                  // ||||       1: Base timer count start
                  // ++++------ Nothing is assigned
}

/************************************************************************/
/*                                                                      */
/* Subroutine:  SWD_HDLC1_Init                                          */
/* Purpose:     Initialization of HDLC1 block of Intelligent I/O group 1  */
/*              General init of receive and transmit part               */
/* Parameter:   No                                                      */
/* Return:      No                                                      */
/*                                                                      */
/************************************************************************/
void SWD_HDLC1_Init (void)
{
   SWD_HDLC1_Basetimer_Init();

   /* Set SIO special communication mode */
   // Set HDLC special communication mode
   G1MR = 0x03;     // XXXX XXXX
                    // |||| ||++- special communication mode
                    // |||| ||    BRG count source select bit
                    // |||| ||    00: output compare
                    // |||| ||    01: SIO
                    // |||| ||    10: BEAN
                    // |||| ||    11: HDLC
                    // |||| |+--- CKDIR
                    // |||| |     0: internal clock
                    // |||| |     1: external clock
                    // |||| +---- STPS (stop bits in UART mode)
                    // ||||       0: 1 stop bit
                    // ||||       1: 2 stop bits
                    // |||+------ PRY (UART parity)
                    // |||        0: Odd parity
                    // |||        1: Even parity
                    // ||+------- PRYE (UART parity enable bit)
                    // ||         0: disable
                    // ||         1: enable
                    // |+-------- UFORM (transfer direction select bit)
                    // |          0: LSB first
                    // |          1: MSB first
                    // +--------- IRS (transmit IR cause select)
                    //            0: transmit buffer empty
                    //            1: transmit shift register empty

   /* Set communication control register */
   // Disable transmit and receice block
   G1CR &= ~0x30;   // XXXX XXXX
                    // |||| |||+- TI (Transmit Buffer empty Flag)
                    // |||| |||   0: data in transmit buffer register
                    // |||| |||   1: no data present in transmit buffer register
                    // |||| ||+-- TXEPT (Transmit register empty flag)
                    // |||| ||    0: data present in transmit register
                    // |||| ||    1: no data present in transmit register
                    // |||| |+--- RI (Receive complete flag)
                    // |||| |     0: no data present in receive buffer register
                    // |||| |     1: data present in receive buffer register
                    // |||| +---- nothing is assigned, when write set to 0
                    // |||+------ TE (Transmit enable bit)
                    // |||        0: Transmission disabled
                    // |||        1: Transmission enable
                    // ||+------- RE (Receive enable bit)
                    // ||         0: Reception disabled
                    // ||         1: Reception enabled
                    // |+-------- IPOL (Receive input polarity reverse select bit)
                    // |          0: no reverse
                    // |          1: reverse
```

```
                // +--------- OPOL (Transmit output polarity reverse select bit)
                //            0: no reverse
                //            1: reverse

/* Set communication control register */
// Enable receice block
G1CR |= 0x20;    // XXXX XXXX
                // |||| |||+- TI (Transmit Buffer empty Flag)
                // |||| |||   0: data in transmit buffer register
                // |||| |||   1: no data present in transmit buffer register
                // |||| ||+-- TXEPT (Transmit register empty flag)
                // |||| ||    0: data present in transmit register
                // |||| ||    1: no data present in transmit register
                // |||| |+--- RI (Receive complete flag)
                // |||| |     0: no data present in receive buffer register
                // |||| |     1: data present in receive buffer register
                // |||| +---- nothing is assigned, when write set to 0
                // |||+------ TE (Transmit enable bit)
                // |||         0: Transmission disabled
                // |||         1: Transmission enable
                // ||+------- RE (Receive enable bit)
                // ||          0: Reception disabled
                // ||          1: Reception enabled
                // |+-------- IPOL (Receive input polarity reverse select bit)
                // |           0: no reverse
                // |           1: reverse
                // +--------- OPOL (Transmit output polarity reverse select bit)
                //            0: no reverse
                //            1: reverse

/* Set function expand receive control register */
// Enable flag detection, bit enstuffing, receive CRC and switch on receive switch
G1ERC = 0xB8;    // XXXX XXXX
                // |||| |||+- CMP0E (compare 0 trigger enable) for Abort detection
                // |||| |||   0: disable
                // |||| |||   1: enable
                // |||| ||+-- CMP1E (comapre 1 enable)
                // |||| ||    0: disable
                // |||| ||    1: enable
                // |||| |+--- CMP2E (compare 2 enable)
                // |||| |     0: disable
                // |||| |     1: enable
                // |||| +---- CMP3E (compare 3 enable) for Flag detection
                // ||||       0: disable
                // ||||       1: enable
                // |||+------ RCRCE (receive CRC enable)
                // |||         0: disable
                // |||         1: enable
                // ||+------- RSHTE (receive shift switch)
                // ||          0: switched off
                // ||          1: switched on
                // |+-------- RBSF0 (bit enstuffing 1 deletion)
                // |           0: disable
                // |           1: enable
                // +--------- RBSF1 (bit enstuffing 0 deletion)
                //            0: disable
                //            1: enable

/* Set compare register values */
G1CMP0 = SWD_HDLC1_ABORT;       // Group 1 data compare register 0 set to 0xFE
G1MSK0 = SWD_HDLC1_ABORT_MASK;  // Group 1 data mask register 0 set to 0x01
G1CMP3 = SWD_HDLC1_FLAG;        // Group 1 data compare register 3 set to 0x7E

/* Set function expand transmit control register */
G1ETC = 0x00;    // XXXX XXXX
                // |||| +---- SOF (SOF transmit request bit)
                // ||||       0: No SOF transmit request
                // ||||       1: SOF transmit request
                // |||+------ TCRCE (transmit CRC enable)
                // |||         0: disable
                // |||         1: enable
                // ||+------- ABTE (arbitration enable bit)
                // ||          0: disable
                // ||          1: enable
                // |+-------- TBSF0 (bit stuffing 1 insertion)
```

```
                    // |          0: disable
                    // |          1: enable
                    // +--------- TBSF1 (bit stuffing 0 insertion)
                    //            0: disable
                    //            1: enable

    /* Set interrupt priority level for Intelligent I/O group1 to zero */
    IIO2IC &= ~0x07;    // For Intelligent I/O group0 receive
    IIO3IC &= ~0x07;    // For Intelligent I/O group0 transmit
    IIO4IC &= ~0x07;    // For compare trigger

    /* Set function expand mode register */
    // Enable auto CRC init and set 0xFFFF as init value
    // Set CRC polynom, select parallel output register
    G1EMR = 0xE6;       // XXXX XXXX
                        // |||| |||+- SMODE (used for BEAN)
                        // |||| |||   0: Normal mode
                        // |||| |||   1: Resynchronous mode
                        // |||| ||+-- CRCV (CRC initialize value bit)
                        // |||| ||    0: initialize 0x0000
                        // |||| ||    1: initialize 0xFFFF
                        // |||| |+--- ACRC (CRC initialization select bit)
                        // |||| |     0: no auto init
                        // |||| |     1: auto init: bit stuffing and CRC are initialized by cmp3 trigger
                        // |||| +---- BSINT: enable bit stuffing error interrupt select bit
                        // ||||       0: disable
                        // ||||       1: enable
                        // |||+------ RXSL (receive root select bit)
                        // |||        0: RxD (BEAN)
                        // |||        1: Receive Input Buffer (HDLC)
                        // ||+------- TXSL (transmit destination select bit)
                        // ||         0: TxD (BEAN)
                        // ||         1: Transmit Output Register (HDLC)
                        // ++-------- CRC0, CRC1 (CRC polynom selection)
                        //            00: x8+x4+x+1 (BEAN)
                        //            01: invalid
                        //            10: x16+x15+x2+1
                        //            11: x16+x12+x5+1 (HDLC)

    /* Set dummy data to receive input buffer of II/O group1 */
    G1RI = 0xFF;

    // Wait for a basetimer clock cycle, before input select is switched to register input
    while ( G1BT >= 0x0010);

    /* Set function expand mode register */
    // Select parallel input register
    G1EMR = 0xF6;       // XXXX XXXX
                        // |||| |||+- SMODE (used for BEAN)
                        // |||| |||   0: Normal mode
                        // |||| |||   1: Resynchronous mode
                        // |||| ||+-- CRCV (CRC initialize bit)
                        // |||| ||    0: initialize 0x0000
                        // |||| ||    1: initialize 0xFFFF
                        // |||| |+--- ACLR (auto clear bit)
                        // |||| |     0: no auto clear
                        // |||| |     1: auto clear: bit stuffing and CRC are initialized by cmp3 trigger
                        // |||| +---- BS_INT: enable bit stuffing error trigger
                        // ||||       0: disable
                        // ||||       1: enable
                        // |||+------ RXSL (receive root select bit)
                        // |||        0: RxD (BEAN)
                        // |||        1: Receive Input Buffer (HDLC)
                        // ||+------- TXSL (transmit destination select bit)
                        // ||         0: TxD (BEAN)
                        // ||         1: Transmit Output Register (HDLC)
                        // ++-------- CRC0, CRC1 (CRC polynom selection)
                        //            00: x8+x4+x+1 (BEAN)
                        //            01: x12+x11+x3+x2+1
                        //            10: x16+x15+x2+1
                        //            11: x16+x12+x5+1 (HDLC)

    // Initialization of all error and event counter
    ulSWD_HDLC1_CntRcvCRCError = ZERO;
    ulSWD_HDLC1_CntRcvBufferOverRun = ZERO;
```

```
   ulSWD_HDLC1_CntRcvCRCFrameOK = ZERO;
   ulSWD_HDLC1_CntRcvAbortError = ZERO;
   ulSWD_HDLC1_CntRcvFrameShort = ZERO;
   ulSWD_HDLC1_CntRcvG1RI_NotEmpty = ZERO;
   ulSWD_HDLC1_CntSndFrameStarted = ZERO;
   ulSWD_HDLC1_CntSndFrameOK = ZERO;
   ulSWD_HDLC1_CntSndG1TO_NotReady = ZERO;
   ulSWD_HDLC1_CntSndG1TB_NotReady = ZERO;

   msgSWD_HDLC1.TransmitStart = SWD_HDLC1_NO; // Set message, that transmission is finished
   msgSWD_HDLC1.ReceiveFlagDetected = SWD_HDLC1_NO; // Set message that no flag has been detected

   ulSWD_HDLC1_RcvIndex = ZERO;  // Init receive HDLC data index
}

/************************************************************************/
/*                                                                      */
/* Subroutine:  SWD_HDLC1_RcvPoll                                       */
/* Purpose:     Routine, which have to be polled in application software */
/*              This routine takes care about the needed action, if the */
/*              received data is equal to HDLC specific pattern, e.g.   */
/*              StartFlag or Abort.                                     */
/*                                                                      */
/* Parameter:   No                                                      */
/* Return:      No                                                      */
/*                                                                      */
/************************************************************************/
void SWD_HDLC1_RcvPoll(void)
{
 if(G1IRF & 0x80)        // Has a start/end flag been detection by Compare Register?
                         // Therefore check compare match register 3 flag at
                         // special communication interrupt detect register
 {
   ucSWD_HDLC1_DummyRead = G1RB;     // Dummy read out of receive output buffer,
   IIO2IR &= ~0x20;                  // Reset interrupt request for receive output buffer full

   G1IRF &= ~0x80;    // Reset start or end flag detection interrupt request

   /* set function expand receive control register */
   // Enable Abort and flag detection, bit enstuffing 0 enable, receive shift on, CRC enable
   G1ERC = 0xB9;

   // If received number of bytes not enough for a complete frame
   if(ulSWD_HDLC1_RcvIndex < SWD_HDLC1_FRAME_MIN)
   {
     ulSWD_HDLC1_RcvIndex = ZERO;     // Reset receive index counter
     ulSWD_HDLC1_CntRcvFrameShort++;  // Increment receive frame to short error counter
     msgSWD_HDLC1.ReceiveFlagDetected = SWD_HDLC1_YES;  // Set message that a flag has been detected
   }

   else if(msgSWD_HDLC1.ReceiveFlagDetected == SWD_HDLC1_NO)
   {
     msgSWD_HDLC1.ReceiveFlagDetected = SWD_HDLC1_YES;  // Set message that a flag has been detected
   }

   else if(msgSWD_HDLC1.ReceiveFlagDetected == SWD_HDLC1_YES)
   {
     CRC_HDLC1_Rcv.word = G1RCRC;   // Read out crc result

     if(CRC_HDLC1_Rcv.word != SWD_HDLC1_GOOD_CRC)  // A wrong CRC has been generated
     {
       ulSWD_HDLC1_RcvIndex = ZERO;   // Reset receive index counter
       ulSWD_HDLC1_CntRcvCRCError++;  // Increment receive CRC error counter
     }

     else  // CRC has been generated correct
     {
       // ----------------------------------------
       // -------- Handle for correct CRC ---------
       // ----------------------------------------
       // Please add your code here for received HDLC frame here
       ulSWD_HDLC1_RcvIndex = ZERO;     // Reset receive index counter
       ulSWD_HDLC1_CntRcvCRCFrameOK++;  // Increment receive CRC OK counter
     }
   } // ENDOF else if(msgSWD_HDLC1.ReceiveFlagDetected == SWD_HDLC1_YES)
```

```
 } //ENDOF if(G1IRF & 0x80)

 else if(G1IRF & 0x10) // Has a abort flag been detection by Compare Register?
                       // Therefore check compare match register 0 flag at
                       // special communication interrupt detect register
 {
   ucSWD_HDLC1_DummyRead = G1RB;  // Dummy read out of receive output buffer,
   IIO2IR &= ~0x20;               // Reset interrupt request for receive output buffer full

   G1IRF &= ~0x10;  // Reset Abort Flag detection interrupt request

   /* set function expand receive control register */
   // Disable Abort and enable flag detection, bit enstuffing 0 enable, receive shift on, CRC enable
   G1ERC = 0xB8;

   msgSWD_HDLC1.ReceiveFlagDetected = SWD_HDLC1_NO; // Set message that no flag has been detected
   ulSWD_HDLC1_RcvIndex = ZERO;           // Reset receive index counter
   ulSWD_HDLC1_CntRcvAbortError++;        // Increment Abort flag counter
 } //ENDOF else if(G1IRF & 0x10)
}

/************************************************************************/
/*                                                                      */
/* Subroutine:  SWD_HDLC1_RcvIn                                         */
/* Purpose:     Routine, which have to be polled in application software  */
/*              Within this routine received HDLC data will be handled,   */
/*              and routed to the received input buffer register.         */
/*                                                                      */
/* Parameter:   No                                                      */
/* Return:      No                                                      */
/*                                                                      */
/************************************************************************/
void SWD_HDLC1_RcvIn(unsigned char * pucRcvIn, unsigned char ucSlot)
{
  if (IIO2IR & 0x10) // Interrupt requested flag is set for G1RI register
  {
    IIO2IR &= ~0x10;  // Reset interrupt request flag
    G1RI = * (pucRcvIn + ucSlot); // Write received data byte in HDLC block
                                                 // receive input buffer
  }
  else
    ulSWD_HDLC1_CntRcvG1RI_NotEmpty++; // Increment G1RI not empty counter
}

/************************************************************************/
/*                                                                      */
/* Subroutine:  SWD_HDLC1_RcvOut                                        */
/* Purpose:     Routine, which have to be polled in application software  */
/*              Within this routine received HDLC data will be handled,   */
/*              and routed to the received output buffer array.           */
/*                                                                      */
/* Parameter:   No                                                      */
/* Return:      No                                                      */
/*                                                                      */
/************************************************************************/
void SWD_HDLC1_RcvOut(void)
{
  if (IIO2IR & 0x20)    // Interrupt has been requested by G1RB register
  {
    IIO2IR &= ~0x20;    // Reset interrupt request flag
    ucHDLC1_RcvOut[ulSWD_HDLC1_RcvIndex] = G1RB; // Read receive output
                                                 // buffer value and store
                                                 // in buffer
    ulSWD_HDLC1_RcvIndex++;  // Increment received bytes index (for current frame)
    if (ulSWD_HDLC1_RcvIndex >= BUFFER_SIZE-1) // Received frame is too long to store in buffer?
    {
       ulSWD_HDLC1_RcvIndex = BUFFER_SIZE-1;   // Set index to max value
       ulSWD_HDLC1_CntRcvBufferOverRun++;      // Increment buffer overrun counter
    }
  }
}

/************************************************************************/
/*                                                                      */
/* Subroutine:  SWD_HDLC1_SndIn                                         */
```

```
/* Purpose:     This function starts the start/end flag, stuffing and CRC */
/*              procedure. To do so, a pointer to an unprocessed frame    */
/*              and the number of bytes of the frame are needed as input  */
/*              parameters for this function.                             */
/*              If this function is called with a new frame as parameter, */
/*              before the previous one has been processed complete, an   */
/*              error flag will be returned.                              */
/*              If start procedure was successful, an OK flag will be     */
/*              returned.                                                 */
/*                                                                        */
/* Parameter:#1 unsigned char * pSndInput - pointer to the buffer where   */
/*              the unprocessed frame is stored                           */
/*           #2 unsigned short usLength_SndInput - length of the frame    */
/* Return:      unsigned char - Error flag, if function is called twice   */
/*                              OK flag, if process has been started      */
/*                                                                        */
/**************************************************************************/
unsigned char SWD_HDLC1_SndIn (unsigned char * pucSndInput,
                               unsigned short usLength_SndInput)
{
   // To check whether Start routine have been entered twice, before previous frame
   // has been processed complete
   if (msgSWD_HDLC1.TransmitStart == SWD_HDLC1_YES)
   {
     return(SWD_HDLC1_ERROR);      // Left function with error feedback
   }

   msgSWD_HDLC1.TransmitStart = SWD_HDLC1_YES;  // Set message, that transmission have been started now

   /* Set communication control register */
   // Enable transmit part
   G1CR |= 0x10;      // XXXX XXXX
                      // |||| |||+- TI (Transmit Buffer empty Flag)
                      // |||| |||   0: data in transmit buffer register
                      // |||| |||   1: no data present in transmit buffer register
                      // |||| ||+-- TXEPT (Transmit register empty flag)
                      // |||| ||    0: data present in transmit register
                      // |||| ||    1: no data present in transmit register
                      // |||| |+--- RI (Receive complete flag)
                      // |||| |     0: no data present in receive buffer register
                      // |||| |     1: data present in receive buffer register
                      // |||| +---- nothing is assigned, when write set to 0
                      // |||+------ TE (Transmit enable bit)
                      // |||        0: Transmission disabled
                      // |||        1: Transmission enable
                      // ||+------- RE (Receive enable bit)
                      // ||         0: Reception disabled
                      // ||         1: Reception enabled
                      // |+-------- IPOL (Receive input polarity reverse select bit)
                      // |          0: no reverse
                      // |          1: reverse
                      // +--------- OPOL (Transmit output polarity reverse select bit)
                      //            0: no reverse
                      //            1: reverse

   pucSWD_HDLC1_SndInput = pucSndInput;      // Set pointer to unprocessed frame
                                          // buffer based on function call parameter
   ucSWD_HDLC1_SndState = SWD_HDLC_SND_STATE_DATA; // Initialize transmit state machine
   usSWD_HDLC1_SndIndex = ZERO;                    // Initialize send index with 0
   usSWD_HDLC1_SndLength = usLength_SndInput; // Initialize transmit length based
                                              // on function call parameter, minus
                                              // one becaus following index
                                              // usSWD_HDLC1_SndIndex starts at zero

   G1TB = SWD_HDLC1_FLAG;   // Transfer start flag to transmit input buffer,
                            // which actually starts the transfer process

   CRC_HDLC1_Snd.word = SWD_HDLC1_CRC_INIT; // Init of backup CRC, respectively Standard CRC

   ulSWD_HDLC1_CntSndFrameStarted++;  // Increment counter of transmitted frames (just started)

   return(SWD_HDLC1_OK);
}

/**************************************************************************/
```

```
/*                                                                      */
/* Subroutine:  SWD_HDLC1_SndOut                                        */
/* Purpose:     Routine, which have to be polled in application software. */
/*              Within this routine transmit HDLC data will be handled,  */
/*              and routed to the specific address, assigned by the      */
/*              parameter.                                              */
/*                                                                      */
/* Parameter:#1 unsigned char *pucSndOut - pointer to the GCI frame     */
/*           #2 unsigned char ucSlot - index parameter for GCI frame    */
/* Return:      No                                                      */
/*                                                                      */
/************************************************************************/
void SWD_HDLC1_SndOut(unsigned char * pucSndOut, unsigned char ucSlot)
{
  if (IIO3IR & 0x10) // Interrupt requested flag is set for G1TO register
  {
    IIO3IR &= ~0x10;    // Reset interrupt request flag
    * (pucSndOut + ucSlot) = G1TO;  // Transfer HDLC transmit output buffer
                                    // to GCI frame
  }
  else // If no data transfer is currently performed
  {
    * (pucSndOut + ucSlot) = SWD_HDLC1_PAUSE; // Transfer HDLC pause data to GCI buffer
    ulSWD_HDLC1_CntSndG1TO_NotReady++;  // Increment G1TO not ready counter
  }
}

/************************************************************************/
/*                                                                      */
/* Subroutine:  SWD_HDLC1_SndPoll                                       */
/* Purpose:     Routine, which have to be polled in application software. */
/*              The statemachine takes care about stuffing, CRC, end flag */
/*              generation                                              */
/*                                                                      */
/* Parameter:   No                                                      */
/* Return:      No                                                      */
/*                                                                      */
/************************************************************************/
void SWD_HDLC1_SndPoll(void)
{
  unsigned char ucDummy;

  if(IIO3IR & 0x20)   // Confirm interrupt request of transmit input buffer (G1TB)
  {
    IIO3IR &= ~0x20;  // Reset interrupt request flag

    if (msgSWD_HDLC1.TransmitStart == SWD_HDLC1_YES)  // Check message, whether transmission has been
                                                      // started
    {
      switch (ucSWD_HDLC1_SndState)         // Switch by transmit state machine
      {
        case SWD_HDLC_SND_STATE_DATA:
            G1ETC |= 0x80;                                      // Enable bit stuffing
            G1TB = *(pucSWD_HDLC1_SndInput+usSWD_HDLC1_SndIndex);// Transfer byte to transmit input
                                                                // register
            CRCD = CRC_HDLC1_Snd.word;                         // Use backup value as new init value
            CRCIN = *(pucSWD_HDLC1_SndInput+usSWD_HDLC1_SndIndex); // Generate new CRC
            asm("NOP");                                         // wait two cycles
            asm("NOP");
            CRC_HDLC1_Snd.word = CRCD;                          // Backup CRC again
            usSWD_HDLC1_SndIndex++;                             // Increment index counter
            if(usSWD_HDLC1_SndIndex >= usSWD_HDLC1_SndLength) // Compare index counter with actually
                                                              // number of bytes, which is reached.
            {                                                //(Actually one byte before end of frame)
              ucSWD_HDLC1_SndState = SWD_HDLC_SND_STATE_CRCL; // If end of frame is reached,
                                                              // change to new state machine state
            }
            break;

        case SWD_HDLC_SND_STATE_CRCL:
            CRC_HDLC1_Snd.word = ~CRC_HDLC1_Snd.word;          // Invert the complete CRC word
            G1TB = CRC_HDLC1_Snd.byte.low;                     // Transfer the low byte of the CRC word
                                                               // to transmit input register
            ucSWD_HDLC1_SndState = SWD_HDLC_SND_STATE_CRCH; // Change to new state machine state
            break;
```

```
        case SWD_HDLC_SND_STATE_CRCH:
            G1TB = CRC_HDLC1_Snd.byte.high;       // Transfer the low byte of the CRC
                                                  // word to transmit input register
            ucSWD_HDLC1_SndState = SWD_HDLC_SND_STATE_FLAG; // Change to new state machine state
            break;

        case SWD_HDLC_SND_STATE_FLAG:
            G1ETC &= ~0x80;                           // Disable bit stuffing
            G1TB = SWD_HDLC1_FLAG;                    // Transfer end flag to transmit input register
            ucSWD_HDLC1_SndState = SWD_HDLC_SND_STATE_FILL;  // Change to new state machine state
            break;

        case SWD_HDLC_SND_STATE_FILL:
            G1MR |= 0x80;              // Change interrupt request trigger to input transmit is completed
            G1TB = SWD_HDLC1_FILL;                // Transfer second fill byte to transmit input register
            ucSWD_HDLC1_SndState = SWD_HDLC_SND_STATE_END;  // Change to new state machine state
            break;

        case SWD_HDLC_SND_STATE_END:
            G1MR &= ~0x80;      // Change interrupt request trigger to transmit input buffer empty (G1TB)
            G1CR &= ~0x10;       // Disable transmit block
            IIO3IR &= ~0x30;    // Reset interrupt request flag for input and output transmit register
            ucDummy = G1TO;      // Dummy read is needed
            msgSWD_HDLC1.TransmitStart = SWD_HDLC1_NO; // Set message, that transmission is finished
            ulSWD_HDLC1_CntSndFrameOK++;
            break;

        default:
            // Wrong state!!!!
            break;

       } // ENDOF switch
      } // ENDOF if(msgSWD_HDLC1.TransmitStart == SWD_HDLC1_YES)
    } // ENDOF if(IIO3IR & 0x20)
    else
       ulSWD_HDLC1_CntSndG1TB_NotReady++;
}

/********************************** E O F ******************************/
```

6 Reference

Renesas Technology Corporation Semiconductor Home Page
http://www.renesas.com

Contact for Renesas Technical Support
E-mail: support_apl@renesas.com

Data Sheet & User's Manual
M32C/83 Datasheet, User's Manual
(Use the latest version, please check: http://www.renesas.com)

Keep safety first in your circuit designs!

- Renesas Technology Corporation puts the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage.
  Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

- These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corporation product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corporation or a third party.
- Renesas Technology Corporation assumes no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
- All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor for the latest product information before purchasing a product listed herein.
  The information described here may contain technical inaccuracies or typographical errors.
  Renesas Technology Corporation assumes no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.
  Please also pay attention to information published by Renesas Technology Corporation by various means, including the Renesas Technology Corporation Semiconductor home page (http://www.renesas.com).
- When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corporation assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
- Renesas Technology Corporation semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
- The prior written approval of Renesas Technology Corporation is necessary to reprint or reproduce in whole or in part these materials.
- If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination. Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
- Please contact Renesas Technology Corporation for further details on these materials or the products contained therein.